



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 Bahnhert 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: 6. Februar 2020

Inhaltsverzeichnis

Abbildungsverzeichnis	V
Listingverzeichnis	VI
1 Einleitung [Pham]	1
2 Das Team mit Namen und Bild	2
3 Ziel des Projekts [Falk]	4
4 Projektplanung	5
4.1 Vorgehensweise [Gentges]	5
4.2 Funktionsumfangs [Falk]	6
4.3 Projektablaufplan [Gentges]	7
4.4 Planung der Software	13
4.4.1 Planung des Mockups [Pham]	13
4.4.2 Planung der Design Patterns [Falk]	16
4.4.3 Planung der Datenstrukturen und Schnittstellen	21
4.4.4 Planung des Presenters [Tom]	23
4.4.5 Planung der View [Tom]	25
4.4.6 Planung der Menüsteuerung [Istogu]	26
4.5 Geplante Aufgabenverteilung im Team	27
5 Beschreibung des Projektverlaufs	29
5.1 Tatsächliche Aufgabenverteilung im Team	29
5.2 Teammeeting-Protokolle	30
5.3 Projekttagebücher	37
5.3.1 Tom Bockhorn	37
5.3.2 Hendrik Falk	39
5.3.3 Dennis Gentges	41
5.3.4 Getuart Istogu	43
5.3.5 Jannis Keienburg	45
5.3.6 Tim Jonas Meinerzhagen	47
5.3.7 Khang Pham	49
5.3.8 Tim Schwenke	51
5.4 Beschreibung von Problemen	53

5.4.1	Programmierkenntnisse [Gentges]	53
5.4.2	Vergleich von Dezimalzahlen	53
5.4.3	Implementierung generischer Kacheln	53
5.4.4	Teamkommunikation in unterschiedlichen Umgebungen [Falk] . .	53
6	Dokumentation der Software	55
6.1	Dokumentation der Paketstruktur des Android-Projektes [Falk]	55
6.2	Dokumentation der View	56
6.2.1	Activities [Bockhorn]	56
6.2.2	Generische Kachelgestaltung [Tom]	57
6.2.3	Kontextbezogene Kacheln [Bockhorn]	57
6.2.4	Kontextfremde Kacheln [Gentges]	59
6.2.5	Kacheltypdefinition durch Enumartion [Pham]	62
6.2.6	Gruppierung der Kacheln im Layout Container [Bockhorn] . . .	63
6.2.7	Implementierung der Menüsteuerung [Istogu]	63
6.3	Dokumentation der Models	66
6.3.1	Operanden [Schwenke]	66
6.3.2	Operationen	69
6.4	Überblick über die Activities der App bzw. der Funktionen	80
6.5	Dokumentation der Navigation zwischen Activities	80
6.6	Dokumentation der Activity-übergreifenden, persistenten Datenhaltung	80
6.7	Dokumentation der programmatischen Beiträge der Teammitglieder . .	80
6.7.1	Tom Bockhorn	80
6.7.2	Hendrik Falk	81
6.7.3	Dennis Gentges	82
6.7.4	Getuart Istogu	83
6.7.5	Jannis Keienburg	84
6.7.6	Tim Jonas Meinerzhagen	85
6.7.7	Khang Pham	86
6.7.8	Tim Schwenke	87
6.8	Beschreibung von Problemen	89
6.8.1	Softwareentwicklung im Team [Schwenke]	89
7	Dokumentation der sonstigen Beiträge der Teammitglieder	96
7.1	Tom Bockhorn	96
7.2	Hendrik Falk	96

7.3	Dennis Gentges	96
7.4	Getuart Istogu	96
7.5	Jannis Keienburg	96
7.6	Tim Jonas Meinerzhagen	96
7.7	Khang Pham	96
7.8	Tim Schwenke	96
8	Fazits aller Teammitglieder	97
8.1	Tom Bockhorn	97
8.2	Hendrik Falk	97
8.3	Dennis Gentges	97
8.4	Getuart Istogu	97
8.5	Jannis Keienburg	97
8.6	Tim Jonas Meinerzhagen	97
8.7	Khang Pham	97
8.8	Tim Schwenke	97
9	Quellenverzeichnis	98
10	Anhang - Quelltext	99
10.1	Model	99
10.1.1	Calculation	99
10.1.2	Operands	155
10.1.3	Settings	171
10.1.4	Stack	180
10.2	View	196
10.2.1	Layout	196
10.2.2	Menu	196
10.2.3	Schemas	197
10.2.4	Sonstiges	197
10.3	Presenter	198
11	Anhang - Verwendeten Tools und Hilfsprogramme	199
	Ehrenwörtliche Erklärung	200

Abbildungsverzeichnis

Abbildung 1: Gruppenfoto	2
Abbildung 2: Tom Bockhorn	3
Abbildung 3: Hendrik Falk	3
Abbildung 4: Dennis Gentges	3
Abbildung 5: Getuart Istogu	3
Abbildung 6: Jannis Keienburg	3
Abbildung 7: Tim Meinerzhagen	3
Abbildung 8: Khang Pham	3
Abbildung 9: Tim Schwenke	3
Abbildung 10: Grundanforderungen Use-Case-Diagramm	7
Abbildung 11: Grobe Darstellung der Funktionen	8
Abbildung 12: Erweitertes Wasserfallmodell	9
Abbildung 13: Projektstrukturplan	10
Abbildung 14: GANTT-Diagramm	12
Abbildung 15: Erster Paper Prototype	14
Abbildung 16: Mid-Fidelity Prototyp	15
Abbildung 17: Mögliche Umsetzung von MVVM	17
Abbildung 18: Mögliche Umsetzung von MVP	18
Abbildung 19: Mögliche Umsetzung von MVC	18
Abbildung 20: Klassendiagramm	21
Abbildung 21: Click Event Handling vom Presenter	24
Abbildung 22: Ablaufdiagramm für die Zuweisung eines Bruches an einer Kachel	26
Abbildung 23: Ordnerstruktur	55
Abbildung 24: Ordnerstruktur Schema	56
Abbildung 25: Listener von Tile	58
Abbildung 26: Grafische Darstellung der Funktion	80
Abbildung 27: Gitflow	90

Listingverzeichnis

Listing 1: Methodenkopf der generischen Schnittstelle	87
Listing 2: Implementierung der generischen Schnittstelle	88
Listing 3: Konzept für Nutzung generischer Schnittstelle	92
Listing 4: Konzept für Nutzung generischer Schnittstelle	94
Listing 5: Action (Schwenke)	99
Listing 6: ArcCosinus (Keienburg)	100
Listing 7: ArcSinus (Keienburg)	101
Listing 8: ArcTangens (Keienburg)	102
Listing 9: CalculationException (Schwenke)	103
Listing 10: Cosinus (Keienburg)	103
Listing 11: Derivation (Keienburg)	104
Listing 12: HighAndLowPoints (Keienburg)	105
Listing 13: Integral (Istogu)	107
Listing 14: IntegralTest (Istogu)	109
Listing 15: Limes (Istogu)	109
Listing 16: LimesTest (Istogu)	111
Listing 17: Logarithm (Keienburg)	112
Listing 18: Logarithm10 (Keienburg)	113
Listing 19: MatrixUtil (Istogu)	114
Listing 20: MatrixUtilTest (Istogu)	116
Listing 21: Minus (Falk)	117
Listing 22: MinusTest (Schwenke)	120
Listing 23: Modulo (Falk)	123
Listing 24: Plus (Schwenke)	124
Listing 25: PlusTest (Schwenke)	127
Listing 26: Power (Falk)	132
Listing 27: Sinus (Keienburg)	134
Listing 28: Slash (Falk)	135
Listing 29: SlashTest (Schwenke)	139
Listing 30: Tangens (Keienburg)	142
Listing 31: Times (Falk)	143
Listing 32: TimesTest (Schwenke)	148
Listing 33: Zeros (Keienburg)	151

Listing 34: Root (Falk)	153
Listing 35: RootTest (Pham)	154
Listing 36: DoubleComparator (Schwenke)	155
Listing 37: DoubleFormatter (Schwenke)	157
Listing 38: Element (Schwenke)	157
Listing 39: ODouble (Schwenke)	158
Listing 40: OEmpty (Meinerzhagen)	159
Listing 41: OFraction (Schwenke)	160
Listing 42: OMatrix (Schwenke)	162
Listing 43: OPolynom (Schwenke)	164
Listing 44: OSet (Schwenke)	166
Listing 45: OTuple (Schwenke)	168
Listing 46: Operand (Schwenke)	171
Listing 47: AllClear (Falk)	171
Listing 48: ClearHistory (Falk)	172
Listing 49: DeleteEntry (Falk)	172
Listing 50: Dot (Falk)	173
Listing 51: Enter (Falk)	173
Listing 52: Inverse (Falk)	174
Listing 53: LoadLayout (Meinerzhagen)	175
Listing 54: SaveLayout (Meinerzhagen)	176
Listing 55: Setting (Falk)	177
Listing 56: Split (Falk)	177
Listing 57: Swap (Falk)	179
Listing 58: TurnAroundSign (Falk)	179
Listing 59: StackInterface (Keienburg)	180
Listing 60: OperandStack (Keienburg)	182
Listing 61: ArcCosinusTest (Keienburg)	185
Listing 62: ArcSinusTest (Keienburg)	186
Listing 63: ArcTangensTest (Keienburg)	186
Listing 64: CosinusTest (Keienburg)	187
Listing 65: DerivationTest (Keienburg)	188
Listing 66: HighAndLowPointsTest (Keienburg)	189
Listing 67: Logarithm10Test (Keienburg)	190
Listing 68: LogarithmTest (Keienburg)	191

Listing 69: SinusTest (Keienburg)	193
Listing 70: TangensTest (Keienburg)	194
Listing 71: ZerosTest (Keienburg)	195
Listing 72: ScreenOrientation	196
Listing 73: StorageLoadingException	196
Listing 74: TileLayout	196
Listing 75: TileLayoutFactory	196
Listing 76: TileLayoutLoader	196
Listing 77: ChooseListMenu	196
Listing 78: DialogMenu	196
Listing 79: InputDouble	196
Listing 80: InputFraction	197
Listing 81: InputMenuFactory	197
Listing 82: InputPolynomial	197
Listing 83: InputTileType	197
Listing 84: ActionTileScheme	197
Listing 85: ErrorTileScheme	197
Listing 86: HistoryTileScheme	197
Listing 87: OperandTileScheme	197
Listing 88: SettingTileScheme	197
Listing 89: StackTileScheme	197
Listing 90: TileScheme	197
Listing 91: Tile	197
Listing 92: TileMapping	198
Listing 93: TileType	198
Listing 94: TypeQuestionable	198
Listing 95: MainActivity	198
Listing 96: Presenter	198

1 Einleitung [Pham]

”Viele Menschen organisieren ihr halbes Leben über Apps. Ohne Apps wüssten viele nicht mehr, wie das Wetter wird, wie der Kontostand lautet, wo der nächste Stau wartet oder wie sich ihr Lieblingsverein gerade schlägt.”¹

Apps gewinnen immer mehr an Bedeutung im Leben der Menschen. 2018 wurden in Deutschland zum ersten Mal mehr als zwei Milliarden Apps heruntergeladen. Damit wurde nicht nur ein neuer Rekord erreicht was die Anzahl an Downloads angeht, sondern auch die erwirtschafteten Umsätze erreichten einen neuen Höchstwert. Zwei Drittel der App-Downloads entfielen hierbei auf den Google Play Store. Auch für die kommenden Jahre wird ein Wachstum für den App-Markt vorausgesagt.²

Aus diesem Grund beschäftigt sich die vorliegende Studienarbeit mit der Entwicklung einer App für Android. Die Entwicklung der App findet hierbei im Rahmen des FHDW Moduls ”Projekte der Wirtschaftsinformatik” statt.

Das Modul Projekte der Wirtschaftsinformatik beinhaltet die Umsetzung eines komplexen Projektes inklusive der Präsentation und der Dokumentation der Projektergebnisse.³ Die vom Team gewählte Aufgabenstellung war hierbei, einen kachelbasierten Taschenrechner nach der umgekehrten, polnischen Notation zu entwickeln.

Dabei soll in der Studienarbeit nicht nur auf die Vorgehensweise der Teammitglieder während der Programmierung der App eingegangen werden. Vielmehr soll in dieser Studienarbeit auf alle Aspekte eingegangen werden, die für die Durchführung eines solchen Projektes benötigt werden. Dies beinhaltet organisatorische Aspekte wie Projekt- und Zeitplanung, die Koordination zwischen den Teammitgliedern und die Dokumentation des Projektes. Für die Durchführung des Projektes standen dem Team dabei etwas mehr als 80 Stunden pro Teammitglied über einen Zeitraum von 5 Monaten zur Verfügung, zusätzlich zu 40 Kontaktstunden in den Vorlesungen, in welchen jedoch hauptsächlich Grundlagenwissen in Python aufgebaut wurde, welches nicht relevant für die vom Team gewählte Aufgabenstellung war.

¹Berg, zitiert nach Rondinella, G. (2019)

²vgl. Rondinella, G. (2019)

³FHDW (2017)

2 Das Team mit Namen und Bild



Abbildung 1: Gruppenfoto



Abbildung 2: Tom Bockhorn

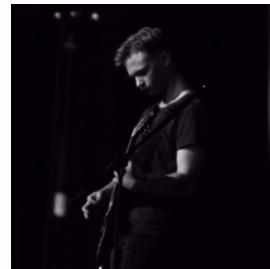


Abbildung 3: Hendrik Falk



Abbildung 4: Dennis Gentges



Abbildung 5: Getuart Istogu



Abbildung 6: Jannis Keienburg

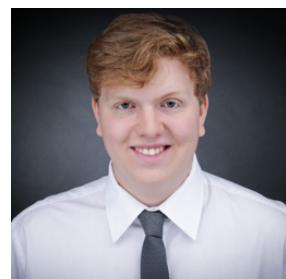


Abbildung 7: Tim Meinerzhagen



Abbildung 8: Khang Pham

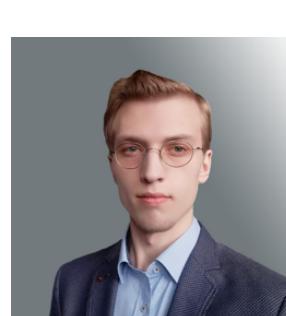


Abbildung 9: Tim Schwenke

3 Ziel des Projekts [Falk]

Das Ziel des Projekts ist die Entwicklung einer kachelbasierten Android-Applikation für den Auftraggeber Prof. Dr. Thomas Seifert. Diese Applikation soll die Funktion eines Taschenrechners nach der umgekehrten, polnischen Notation (UPN) erfüllen und wird im Folgenden als Tile Calculator bezeichnet. Dies stellt die Prüfungsleistung im Modul "Projekte in der Wirtschaftsinformatik" des Teams "Das Proletariat" dar.

Das Team besteht aus folgenden Studierenden der Gruppe BFWI317B und der Fachhochschule der Wirtschaft Bergisch Gladbach: Tom Bockhorn, Hendrik Falk, Dennis Gentges, Getuart Istogu, Jannis Luca Keienburg, Tim Jonas Meinerzhagen, Khang Pham und Tim Schwenke. Diese absolvieren das Wirtschaftsinformatikstudium mit Schwerpunkt IT-Consulting als Angestellte und Auszubildende der Bayer AG, Bayer Business Services GmbH und der Currenta GmbH & Co. OHG.

Der Projektzeitraum erstreckt sich vom 03.09.2019 bis zum 06.02.2020, wobei die Aufteilung der Zeit dem Team selber überlassen war.

Die zu entwickelnde Applikation, sowie der Prozess zur Erstellung selber soll ausführlich dokumentiert werden und zusammen mit der Applikation, welche auf der zur Verfügung gestellten Hardware installiert sein muss, eingereicht werden.

Die folgenden Termine müssen eingehalten werden, damit das Projekt als erfolgreich gilt:

- 05.09.2019: Hochladen der aktuellen Version des Projekttagebuchs und Vorlage beim Dozenten.
- 05.11.2019: Hochladen der aktuellen Version des Projekttagebuchs.
- 08.01.2020: Hochladen der aktuellen Version des Projekttagebuchs.
- 05.02.2020: Hochladen der aktuellen Version des Projekttagebuchs.
- 06.02.2020: Hochladen der Individualversion der Studienarbeit und des Projekts (Deadline: 17:00).
- 08.02.2020: Präsentation des Projektergebnisses und Abgabe der ausgedruckten Team-Version der Studienarbeit sowie des zur Verfügung gestellten Tablets.

Alle hochzuladenden Dateien werden im vom Auftraggeber erstellten Microsoft Teams abgegeben.

4 Projektplanung

4.1 Vorgehensweise [Gentges]

In der Projektplanung sollte die grundlegende Struktur für das weitere Vorgehen festgelegt werden. Zunächst sollte im Kapitel 4.2 der Funktionsumfang beschrieben, um ein einheitliches Verständnis darüber zu bekommen, welche Funktionen die Applikation später erfüllen muss und für welches Betriebssystem diese entwickelt wird.

Im Projektablaufplan (Kapitel 4.3) werden die in der Planung obligatorischen Dokumente, wie z.B. ein Projektstrukturplan oder ein GANTT-Diagramm, sowie das Verfahrensmodell beschrieben.

Im Kapitel 4.4 wird das Planungsvorgehen der Software erläutert, neben der Planung des Mockups wird hierbei ebenfalls auf die Planung des Design Patterns eingegangen.

Anschließend werden die vorliegenden Datenstrukturen und Schnittstellen beschrieben.

Danach wird die persistente Datenhaltung auf das Projekt angewendet und beschrieben.

Zuletzt wird neben der Planung des Presenters, der View sowie der eigentlichen Menüsteuerung eine grobe Übersicht über die geplante Aufgabenverteilung im Team gegeben.

4.2 Funktionsumfangs [Falk]

Im Rahmen dieses Projekts muss eine Android-App entwickelt werden, welche die Funktion eines kachelbasierten UPN-Taschenrechners zur Verfügung stellt.

Dabei müssen drei verschiedene Kacheltypen implementiert werden: *Operator*, *Operand* und *CalculatorFunction*. Ein Operator kann hier zum Beispiel die Wurzel oder Division sein und kann eine beliebige Stelligkeit aufweisen. Ein Beispiel für einen Operanden wären Dezimalzahlen oder Matrizen. Die CalculatorFunction-Kachel kann verschiedene Einstellungen bereitstellen. Dazu zählen u.a. Speicherung eines Layouts, Festlegung des Typs einer Kachel. Jede Kachel muss veränderbar sein und jede Rolle annehmen können. Eine Kachel darf nicht grundsätzlich gesperrt sein.

Die Rechnungen sollen in einer bestimmten Reihenfolge geschehen. Zuerst wählt man die Kachel(n) aus, die den(die) benötigten Operanden beinhalten, dann die Kachel, in welche das Ergebnis geschrieben werden soll und schlussendlich sollte die Operation durchgeführt werden. Die ersten beiden Schritte können unter Umständen entfallen.

Ein grober Überblick über die zu implementierenden Aktionen des Nutzers sind nachfolgend dargestellt.

Während der Bearbeitung des Projekts soll ein besonderes Augenmerk auf drei Aspekte gelegt werden. Der erste ist die komfortable Festlegung von Operanden-Kacheln. Dazu zählt auch das Ändern selbiger und derer Reihenfolge. Der zweite Punkt ist die Konfiguration der Ergebniskacheln. Der dritte Punkt sagt aus, dass die App so benutzerfreundlich wie möglich gestaltet werden soll. Dies soll zum Beispiel durch intuitive Bedienung, minimale Userinteraktion und übersichtliche Gestaltung erreicht werden.

Die App soll für Android in Java entwickelt werden. Prinzipien des objektorientierten Software Entwurfs sollen hierbei eingehalten werden. Während der Durchführung des Projektes soll das Team nach dem erweiterten Wasserfallmodell arbeiten. Nach Möglichkeit sollen etablierte Entwurfsmuster eingesetzt werden. Es soll eine strikte und stark ausgeprägte Modularisierung durchgeführt werden. Hierbei soll die App das Samsung Galaxy Note 8.0 unterstützen. Die Unterstützung weiterer Geräte ist optional. Sowohl der Portrait Modus als auch der Landscape Modus sollen unterstützt werden. Ein Wechsel während der Nutzung der App soll möglich sein. Generell sollen Änderungen der Konfiguration des Androidsystems während der App-Ausführung unterstützt werden.

⁴eigene Darstellung

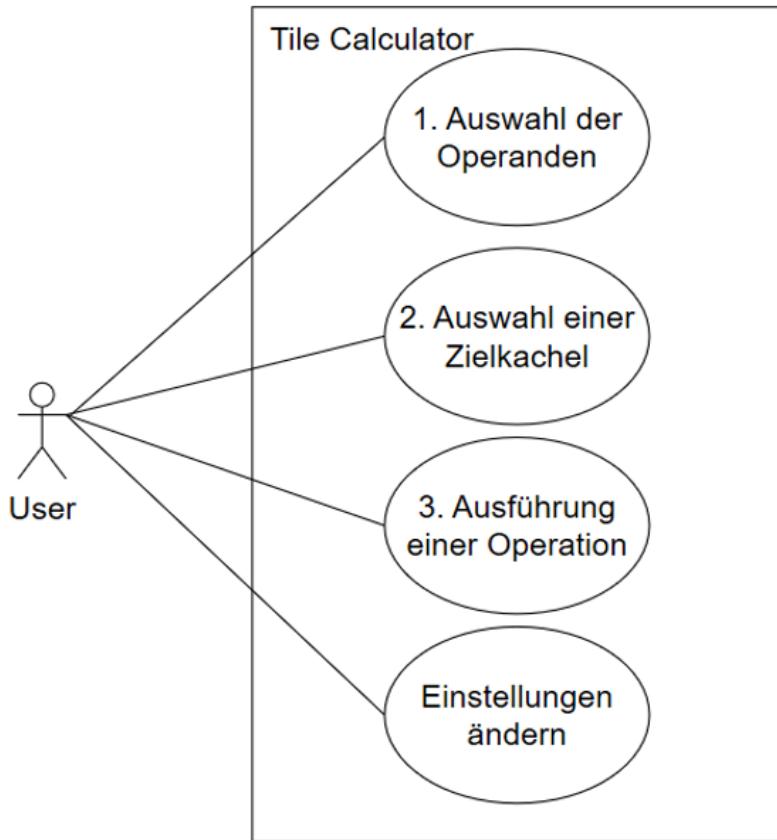


Abbildung 10: Grundanforderungen Use-Case-Diagramm⁴

Das Projekt soll in Android Studio importiert werden können. Der Import soll lokal, das heißt ohne eine bestehende Netzwerkverbindung möglich sein.

4.3 Projektablaufplan [Gentges]

Am Anfang musste ein geeignetes Vorgehensmodell ausgewählt werden. Dabei wählte ich als Projektleiter das erweiterte Wasserfallmodell. Die eindeutigen Argumente für dieses Modell waren zum einen die klare Struktur und der geringe Managementaufwand. Weiterhin war die gute Übersichtlichkeit und eine einfache Verständlichkeit ein klarer Vorteil gegenüber anderen Modellen, wie z.B. SCRUM. Außerdem ist das erweiterte Wasserfallmodell sehr gut geeignet für kleinere Projekte mit genau definiertem Umfang.

Für das vorliegende Projekt wurde das erweiterte Wasserfallmodell mit 8 Phasen aus-

⁵eigene Darstellung

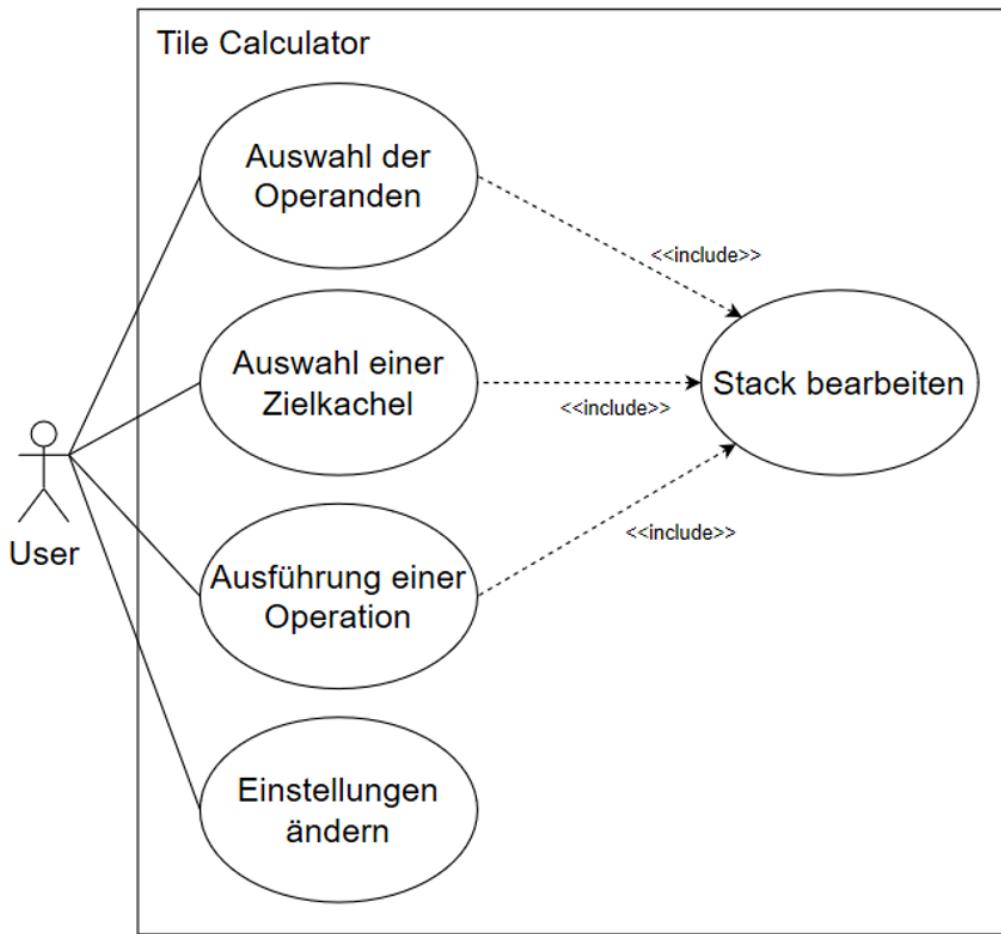


Abbildung 11: Grobe Darstellung der Funktionen⁵

gewählt.

”Es ist ein lineares Vorgehensmodell in der Softwareentwicklung, bei dem der Softwareentwicklungsprozess in einzelnen, festen Phasen organisiert wird. Dabei gelten die Phasenergebnisse immer als bindende Vorgaben für die nächste Phase.”⁷

Bei dem erweiterten Wasserfallmodell sind, begleitend zu der kontinuierlichen Kontrolle des Projektfortschritts, Rücksprünge in die vorherige Bearbeitungsphase möglich, sollte z.B. ein Fehler in einer nachfolgenden Phase identifiziert werden. Das erweiterte Wasserfallmodell bietet eine gute Sicht über den gesamten Projektablauf, da alle Aktivitäten einer Phase vorerst vollständig abgeschlossen werden müssen, bevor man in die

⁶eigene Darstellung

⁷Itemis (2015)

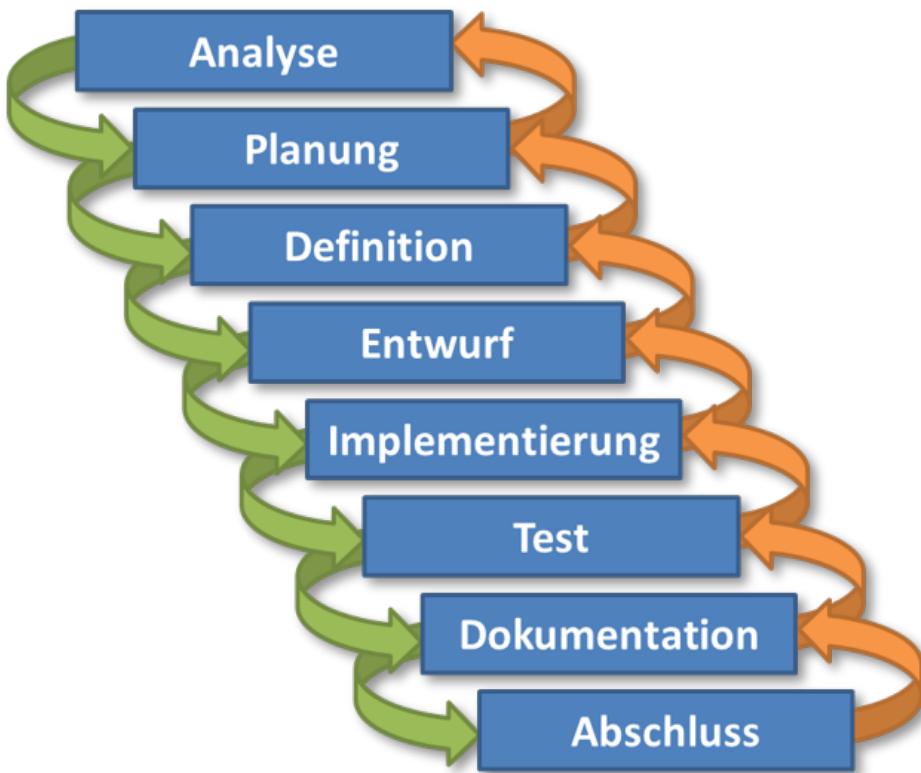


Abbildung 12: Erweitertes Wasserfallmodell⁶

darauffolgende Phase übergeht.

Auf diesem Vorgehensmodell basierte die Projektstrukturplanung, die Meilensteinplanung sowie die Zeitplanung, welche in einem GANTT-Diagramm visualisiert wurde.

Der Projektstrukturplan gliedert sich in mehrere Teilprojekte mit jeweils dazugehörigen Arbeitspaketen.

”Der Projektstrukturplan ist die vollständige Darstellung aller Elemente eines Projektes und ihrer Beziehungen. Dabei werden die Elemente hierarchisch gegliedert, so dass eine Baumstruktur entsteht.“⁸

Den einzelnen Arbeitspaketen und Teilaufgaben sind weitere, kleinere Arbeitspakete und Teilaufgaben zugewiesen.

⁶Windolph, A. (2020)

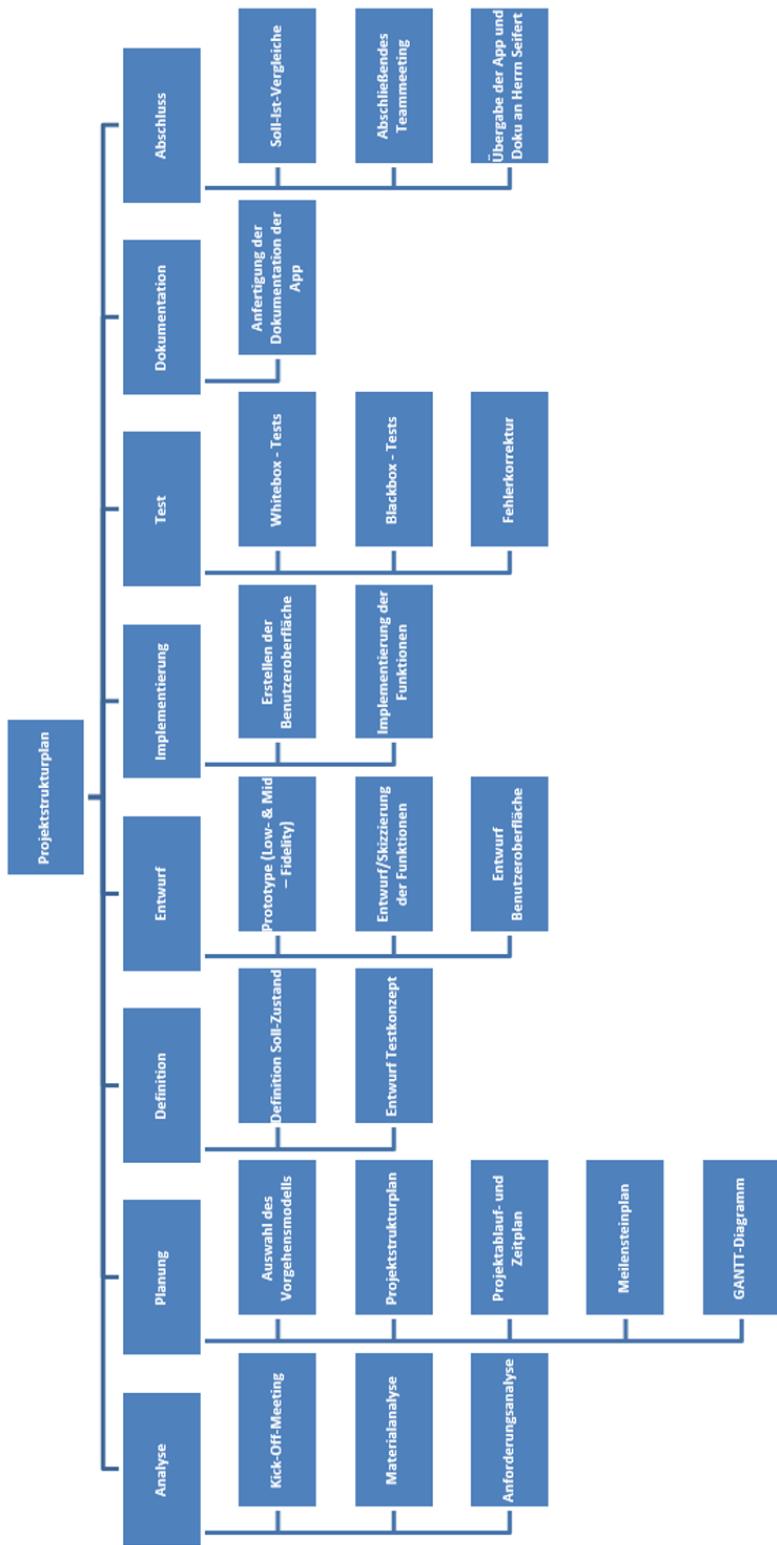


Abbildung 13: Projektstrukturplan⁹

⁹eigene Darstellung

ID	Meilenstein	Datum
1	Kick-Off Meeting abgehalten	03.09.2019
2	Vorgehensmodell ausgewählt	04.09.2019
3	Zeitplanung (GANTT) und Kommunikation	04.09.2019
4	Aufgabenverteilung	05.09.2019
5	Mockups fertiggestellt	04.01.2020
6	Android Studio Implementierung	25.01.2020
7	App getestet	28.01.2020
8	Finale App Version	04.02.2020
9	Dokumentation fertig	05.02.2020
10	Abgabe der Ausarbeitung bei Herrn Seifert	06.02.2020
11	Präsentation	08.02.2020

”Meilensteine gliedern und strukturieren Projekte. Ein Meilensteinplan ist ein häufig genutztes Werkzeug im Projektmanagement und zeigt die Meilensteine eines oder mehrerer Projekte in chronologischer Reihung an.”¹⁰

Der Meilensteinplan wurde dabei unmittelbar nach dem Kick-Off Meeting an das Team kommuniziert. Dabei konnten die Daten der Meilensteine eingehalten werden. Zusammenfassend wird dies in einem Soll-Ist-Vergleich von meinem Kollegen Hendrik Falk näher beleuchtet.

GANTT Diagramme sind im Projektmanagement ein gängiges Tool, um Aktivitäten in Relation mit der Zeit zu setzen. Das Diagramm enthält eine Auflistung von Aktivitäten mit zugewiesenen Zeitleisten.¹¹

¹⁰T2Informatik (2020)

¹¹vgl. Gantt.com (2020)

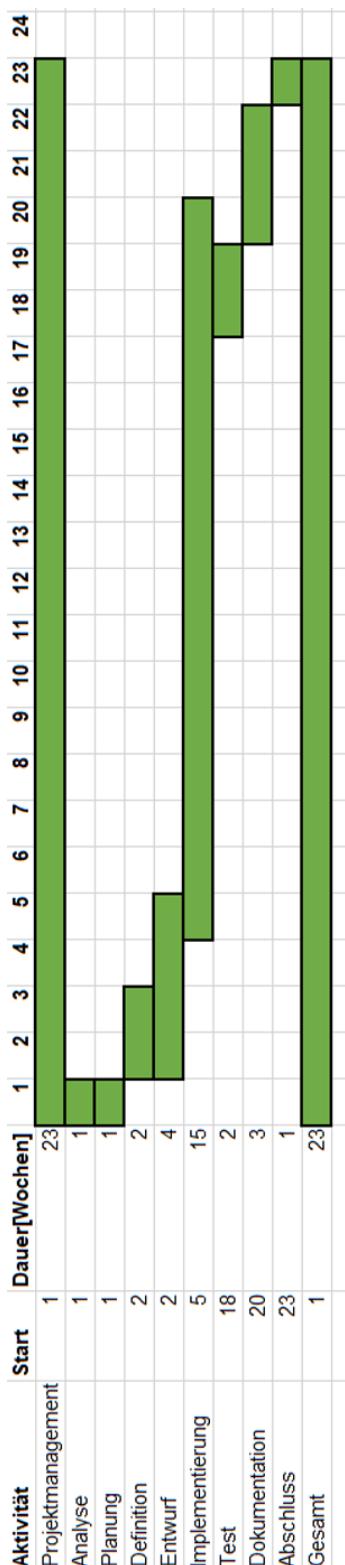


Abbildung 14: GANTT-Diagramm¹²

¹²eigene Darstellung

4.4 Planung der Software

4.4.1 Planung des Mockups [Pham]

Um ein erstes Konzept für eine mögliche Umsetzung des Projektes zu erstellen, wurde zunächst ein Mockup in Form eines Paper-Prototyps erstellt. Bei der Erstellung des Mockups sollten die grundlegenden Regeln des Usability Engineerings eingehalten werden. Unter Usability Engineering ”versteht man den Prozess, der parallel zur klassischen Planungs- und Entwicklungsarbeit die spätere Gebrauchstauglichkeit eines Systems sicherstellt.”¹³ Das heißt, um für eine möglichst hohe Usability des Endproduktes zu sorgen, wurden bereits während der Konzeptionierung des Mockups Usability Aspekte mit in Betracht gezogen.

Um sicherstellen zu können, dass das Konzept für die Umsetzung des Projektes sich mit den Vorstellungen des späteren Benutzers und Auftraggeber (Prof. Dr. Thomas Seifert) deckt, wurde das gegebene Material analysiert sowie ein Gespräch mit dem Benutzer geführt, um die generellen Anforderungen ableiten zu können.

Die spezifischen Anforderungen, die sich während des Interviews ergaben, wurden dabei dokumentiert und mithilfe der gesammelten Anforderungen konnte der erste Low-Fidelity Paper Prototyp erstellt werden.

Ein Low-Fidelity Prototyp dient hierbei lediglich als ein erstes Proof of concept, in dem das grundlegende Konzept und das erste Design festgehalten werden sollen. Eine Funktionalität des Konzeptes ist hierbei nicht notwendig.¹⁵

Auf die genaueren Details des Low-Fidelity Paper Prototyp wird in Kapitel 7.1.5.3 XXX „Erstellung des Low-Fidelity ,Paper Prototyps“ eingegangen. Ziel des Low-Fidelity Paper Prototyps war es, mit möglichst geringem Aufwand so nah wie möglich an die Vorstellungen des Benutzers zu kommen und den Prototypen iterativ zu verbessern. Aus diesem Grund wurde das erste entwickelte Mockup dem Benutzer erneut präsentiert, um Feedback einzuholen und das Konzept iterativ verbessern zu können. Die dabei identifizierten Diskrepanzen zwischen dem Konzept und den Vorstellungen des Benutzers wurden festgehalten und mithilfe des gesammelten Feedbacks konnte ein zweiter Paper Prototyp erstellt werden. Nachdem die Zwischenergebnisse mit dem Benutzer verifiziert wurde, wurde ein Mid-Fidelity Prototype in MS PowerPoint angefertigt.

¹³Usability, H. (2007), S. 204

¹⁴eigene Darstellung

¹⁵vgl. Hammond, J., T. Gross und J. Wesson (2002), S. 204

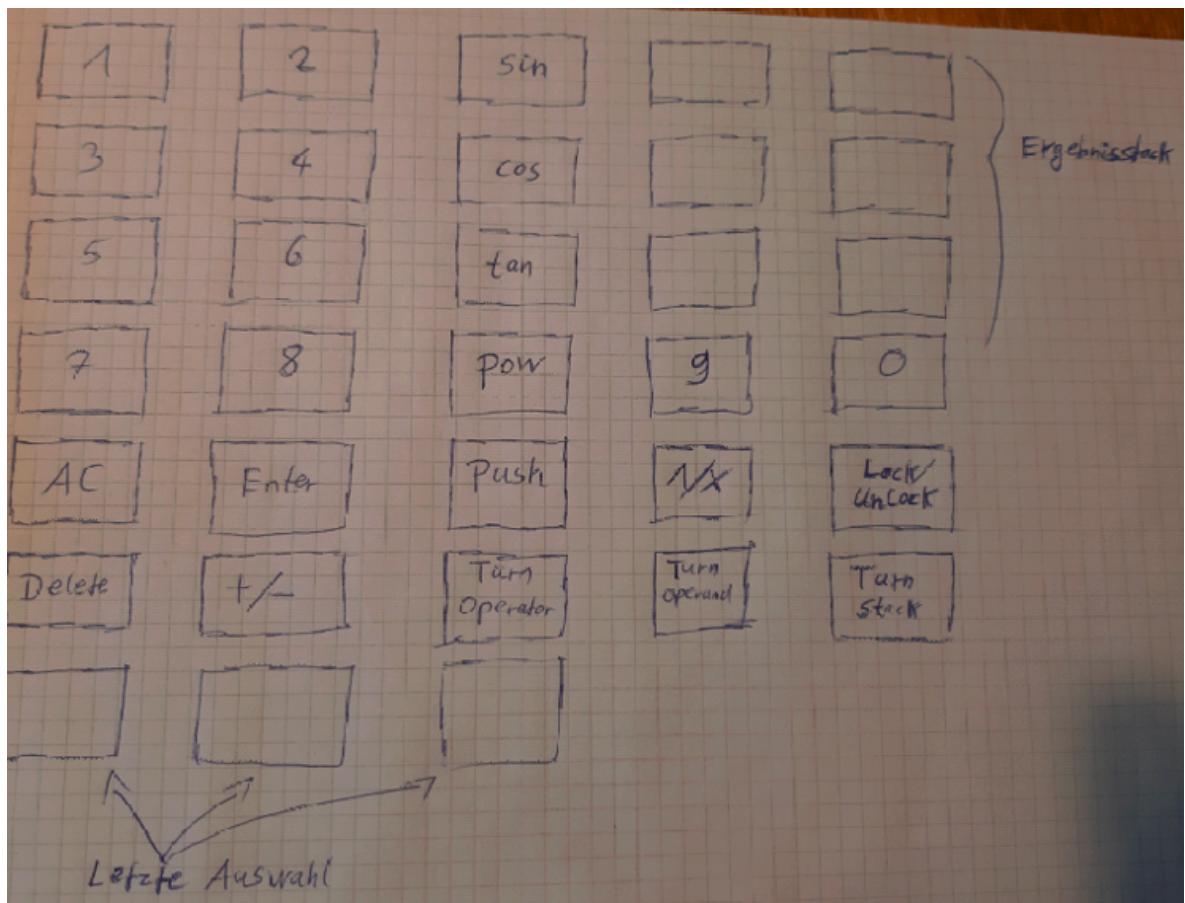


Abbildung 15: Erster Paper Prototype¹⁴

Der Mid-Fidelity-Prototyp ist hierbei interaktiv, relativ detailliert und simuliert das erwünschte Verhalten des finalen Produktes. Es soll im Mid-Fidelity-Prototyp bereits auf Aspekte wie Funktionalität, finales Design oder Navigation eingegangen werden.¹⁶

Im Beispiel unseres Mid-Fidelity Prototyps wurde genauer auf die Umsetzung des Kachelkonzept des Taschenrechners eingegangen ebenso wie die Anforderung für die "Minimierung der Anzahl der für die Ausführung einer Rechnung erforderlichen Interaktionen" mithilfe einer hohen Usability umgesetzt werden sollte. Auf diese Aspekte des Mid-Fidelity Paper Prototyp wird näher in Kapitel XXX 7.1.3.1 „Erstellung der Funktionalitäten des Mid-Fidelity Prototyps“ eingegangen.

Weiterhin wurde ein Standardlayout für den Taschenrechner entworfen und designt, welches beim Aufruf des Taschenrechners angezeigt werden sollte. Detaillierter wird

¹⁶vgl. Hammond, J., T. Gross und J. Wesson (2002), S. 204

¹⁷eigene Darstellung

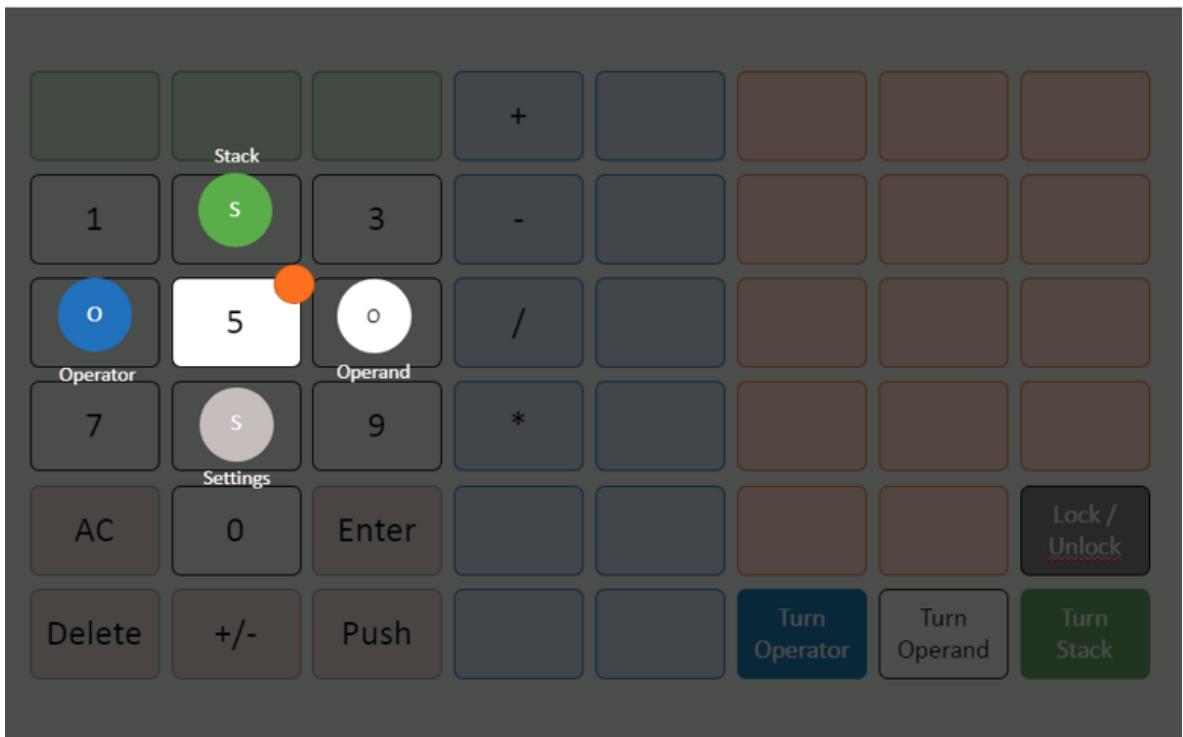


Abbildung 16: Mid-Fidelity Prototyp¹⁷

dieses Design in Kapitel XXX 7.1.7.1 "Erstellung des Design Konzeptes des Mid-Fidelity Prototyps" beschrieben.

Insgesamt ließ sich bei dem Mid-Fidelity-Prototyp das geplante Layout bereits sehr detailliert visualisieren, so dass eine Umsetzung und das finale Design in Android Studio auf Basis des Prototyps bereits möglich gewesen wären. Auch konnten durch die Interaktionsfähigkeit des Prototyps die verschiedenen Use-Cases interaktiv durchgespielt und getestet werden. Nach Fertigstellung des Mid-Fidelity-Prototyps in PowerPoint durch das GUI-Team, wurde das Design und das allgemeine Benutzeroberflächenkonzept des Prototyps dem gesamten Projekt-Team vorgestellt und anschließend vom allen Teammitgliedern in einem der Zwischen-Meetings abgesegnet.

Damit war ein detailliertes Konzept der geplanten Applikation erstellt und der Mid-Fidelity-Prototyp konnte als Basis für die weitere Entwicklung der Applikation verwendet werden.

4.4.2 Planung der Design Patterns [Falk]

Die Teammitglieder suchten einen sinnvollen Ansatz, wie es möglich ist, die Software in verschiedenen Komponenten zu unterteilen und damit sowohl Kollaboration, als auch einfache Verständlichkeit gewährleistet werden können. Design Pattern dienen grundsätzlich der Lösung wiederkehrender Entwurfsprobleme. Außerdem geben sie einer die Möglichkeit, Gebrauch von vielen Best Practices zu machen, z.B. derer der *Gang of Four*.

In diesem Projektabschnitt wurden User Interface Design Pattern betrachtet. Die Anwendung dieser Pattern dient der Trennung von UI Designern, welche sich dadurch auf die Nutzererfahrung und das Aussehen kümmern können, und Back-End-Entwicklern, damit diese sich dann auf die technische Umsetzung konzentrieren können.

Die Teammitglieder betrachteten hierzu verschiedenen MVx Pattern. Dabei steht das "M" für Model, also die tatsächliche Businesslogik der Applikation (tatsächliche Rechenoperationen) und das "V" für View, also die Ansicht, die der Endnutzer sieht. Das "x" steht für ein kommunikatives Bindeglied, was je nach Implementierung variiert. Diese haben ihren Ursprung im Jahre 1979, in welchem Trygve Reenskaug das Model View Controller (MVC)-Pattern entwarf.¹⁸ Aus diesem Pattern entwickelte sich auch das Model View Presenter (MVP)-Pattern. Dieses wurden bereits in den 1990er Jahren eingesetzt, die ausschlaggebende Definition stammt jedoch von Martin Fowler aus dem Jahre 2004.¹⁹ Ein Jahr später, 2005, veröffentlichte John Gossman dann das Model View Viewmodel (MVVM)-Pattern.²⁰

Jede Implementierung weißt eine unterschiedliche Kommunikation zwischen den Komponenten auf. Der Ursprung dessen liegt darin, dass die Komponenten unterschiedliche Wissensstatus aufweisen, also ob die Instanzen der anderen Komponenten übergeben wurden oder nicht.

Die einzelnen Komponenten kommunizieren entweder über direkte Zugriffe auf Attribute und Methoden miteinander oder aber indirekt. Indirekte Kommunikation findet z.B. über die Rückgabewerte Methoden statischer Klassen oder aber über Events durch das Observer Pattern statt. Nach dem Observer Pattern würde Beispielsweise ein Presenter ein Beobachter für das Model, welches das Subjekt wäre. Dabei informiert das Subjekt den Beobachter über seine Veränderungen. Zuerst betrachteten die Teammitglieder

¹⁸vgl. c2-wiki (2014)

¹⁹vgl. Fowler, M. (2006)

²⁰vgl. Roden, G. (2013)

das modernste dieser Pattern, das MVVM-Pattern. Zur Erklärung der Pattern werden Diagramme verwendet, welche eine Mögliche Umsetzung der Pattern darstellen.

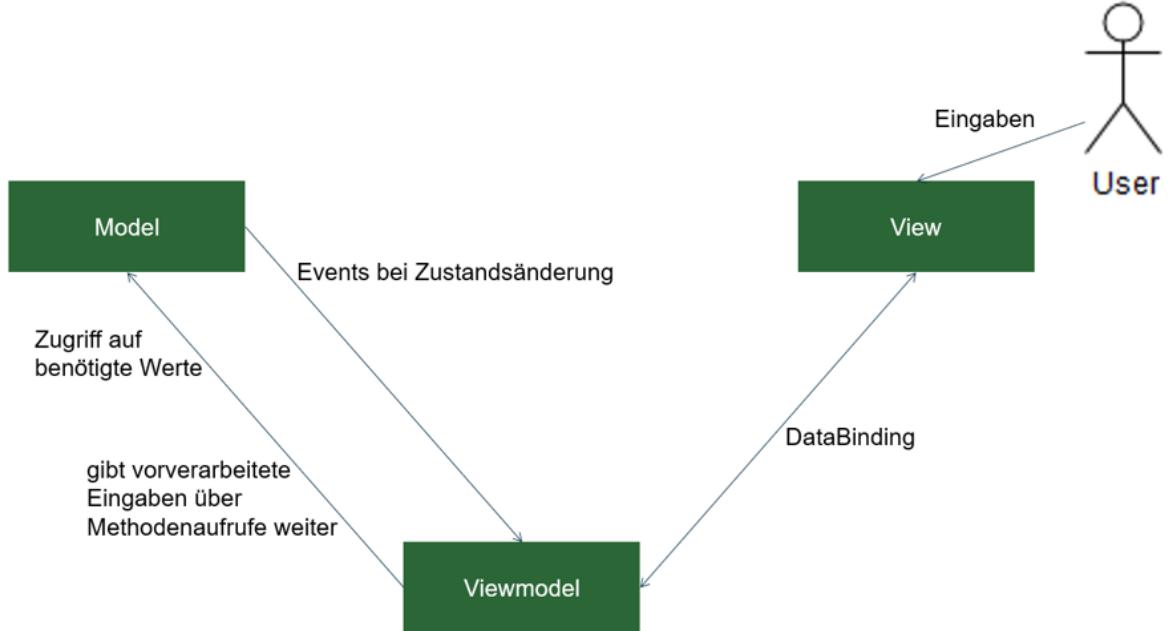


Abbildung 17: Mögliche Umsetzung von MVVM²¹

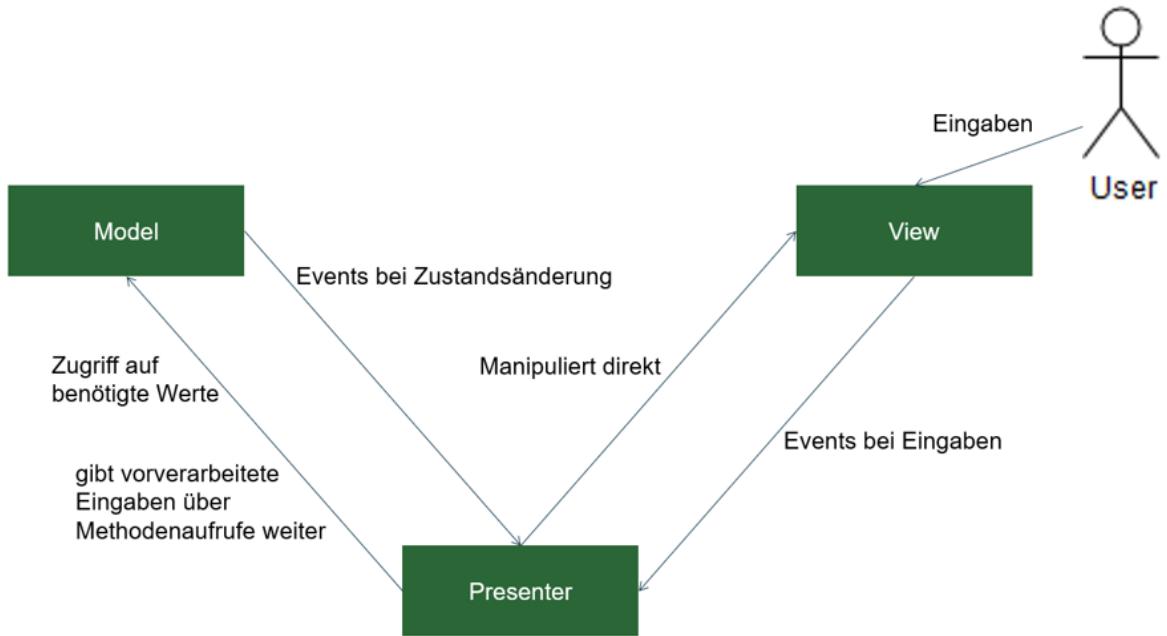
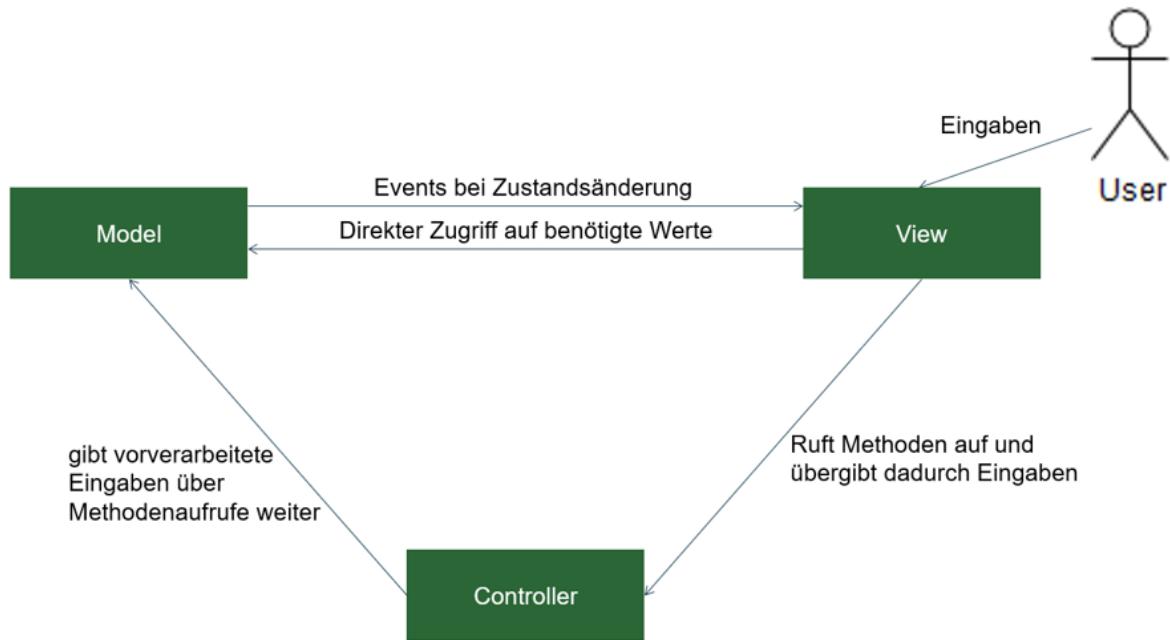
Der User interagiert in jedem Design Pattern mit der View und gibt dort seine Eingaben ein. An diesem Diagramm sieht man, dass bei MVVM die Kommunikation von Model zu Viewmodel über das Observer Pattern funktioniert und das Viewmodel direkt Methode aus dem Model aufruft. Viewmodel und View kommunizieren über DataBinding. Dabei werden Attribute des Viewmodels direkt mit Feldern der View verbunden. Dies führt dazu, dass man in der View lediglich die Anmeldung dieser Felder beim Viewmodel implementiert werden muss. Dieses Pattern kann teilweise bei der Implementierung des Viewmodels sehr kompliziert sein, bietet aber dafür die loseste Kopplung zu der View. Da eine so strenge Trennung für dieses Projekt allerdings nicht benötigt wurde, entschieden sich die Teammitglieder gegen dieses Pattern und betrachteten die nächsten Pattern MVP und MVC.

An den Darstellungen kann man sehen, dass sich die MVP und MVC Pattern sehr stark ähneln. Es unterscheidet sich nur die Kommunikation zwischen Model und View. Während beim MVC Pattern Model und View direkt miteinander kommunizieren, findet

²¹eigene Darstellung

²²eigene Darstellung

²³eigene Darstellung

**Abbildung 18:** Mögliche Umsetzung von MVP²²**Abbildung 19:** Mögliche Umsetzung von MVC²³

dies beim MVP Pattern über den Presenter statt. Aufgrund der hier kleinen Projektgröße war Simplizität ein sehr wichtiges Kriterium bei der Auswahl des UI Design Patterns. Da Android Usereingaben eventgetrieben verarbeitet, hielten die Teammitglieder eine MVP Implementierung für die geeignetste Lösung dieses Problems.

4.4.2.1 Singleton Pattern [Schwenke] XXX

XXX

4.4.2.2 Factory Loader Pattern [Bockhorn] XXX

XXX

4.4.3 Planung der Datenstrukturen und Schnittstellen

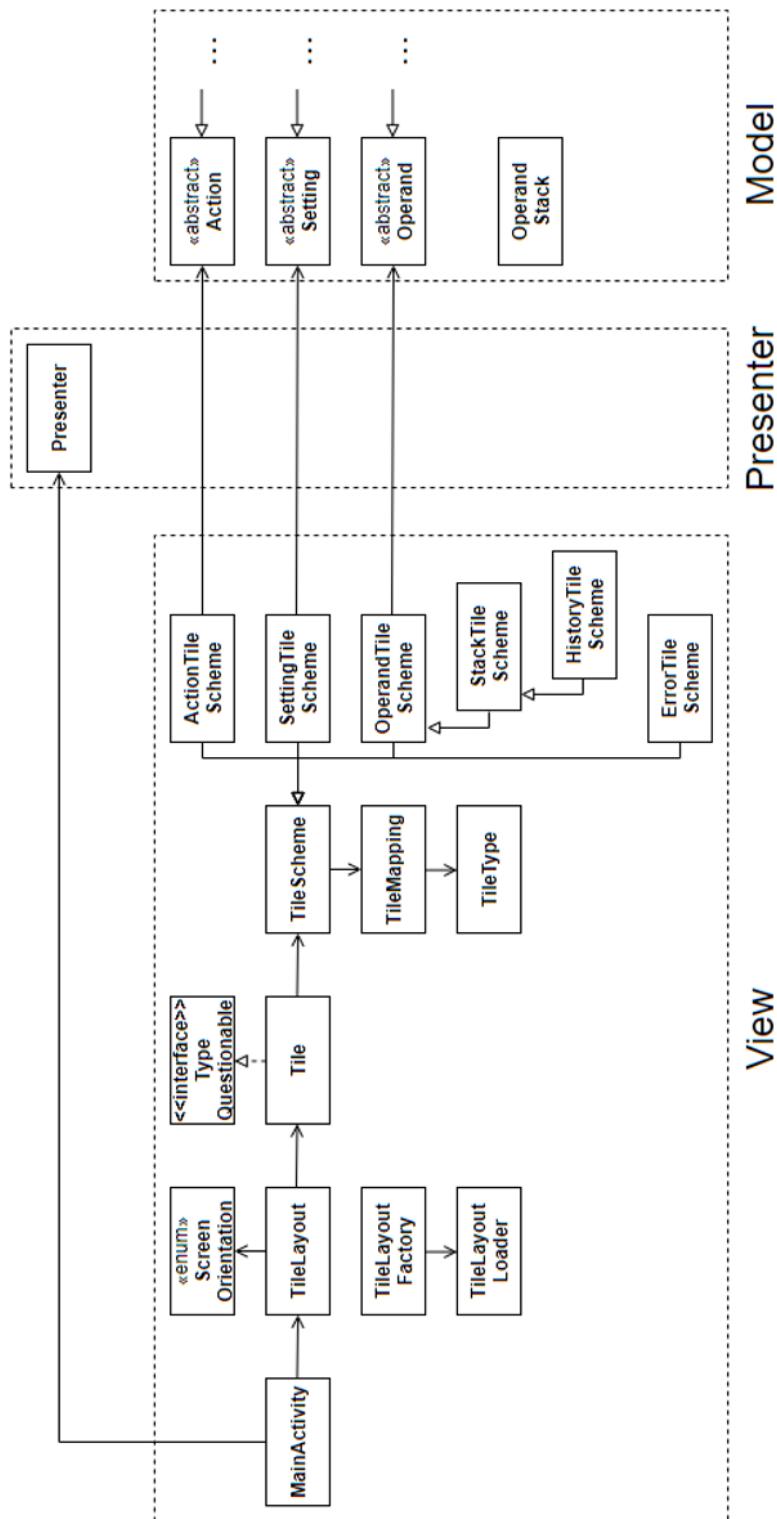


Abbildung 20: Klassendiagramm XXX²⁴

²⁴eigene Darstellung

4.4.3.1 Persistente Datenhaltung²⁵ [Meinerzhagen]

Im Rahmen der persistenten Datenspeicherung wurde zuerst betrachtet, was dauerhaft gespeichert werden muss und welche Elemente beim Start der Anwendung neu generiert werden können. Anschließend wurden die verfügbaren Speichermethoden betrachtet und im Anwendungskontext evaluiert. Die passendste Methode wurde letztlich ausgewählt.

Entsprechend den erarbeiteten Anforderungen wurden in diesem Bereich das Speichern und Laden des aktuellen Zustandes der Kacheln und deren Layout in den Vordergrund gesetzt. Das Layout soll einem Gitternetz entsprechend aufgebaut sein.

Zum Abspeichern dieser Daten wurden die verschiedenen Optionen betrachtet. Android bietet vier Varianten zur persistenten Datenspeicherung nativ an. Diese sind Shared Preferences, interne Speicher, externe Speicher und eine Datenbank. Mit Shared Preferences können einfache Key-Value-Paare gespeichert werden. Für diese wird automatisch eine Datei erstellt, welche Android verwaltet. Der interne Speicher bietet ein, vom restlichen Android System abgekapseltes, Dateisystem. Nur die App selbst ist erlaubt dort Daten zu speichern und lesen. Dagegen beschreibt der externe Speicher das bestehende Dateisystem von Android, auf welches der Nutzer und andere Apps freien Zugriff haben. Letztlich kann eine Datenbank zum Speichern verwendet werden. Android bietet dazu eine SQLite Datenbank an.

Auf Grund der tabellenähnlichen Struktur der zu speichernden Daten wurde zum Speichern ein Tabellenformat angedacht. Dabei fiel die Wahl auf das gängige CSV Format. Mit der Wahl dieses waren die Shared Preferences und die Datenbank nicht mehr nutzbar. Somit lag die Wahl zwischen dem internen und externen Speicher. Da ein Zugriff auf die gespeicherten Daten von außerhalb nicht notwendig ist, fiel die Wahl auf die Nutzung des internen Speichers.

²⁵vgl. Ogbo, O. (2016)

4.4.4 Planung des Presenters [Tom]

XXX

Der Presenter agiert als Bindeglied zwischen der View und dem Model, sodass alle Bereiche ihre Funktionalitäten im vollen Maß erfüllen können und eine Separierung der Anwenderschnittstelle und der Geschäftslogik gewährleistet werden kann. Der Presenter koordiniert die Kommunikation zwischen View und Model, also steuert Rechnungen im Backend und sorgt für die korrekte Darstellung der Ergebnisse in den einzelnen Kacheln der View.

Mit der View kommuniziert er über eine Container Klasse (XXX), welche die einzelnen Kacheln koordiniert. Aufrufe der View in Richtung des Presenters werden durch Events realisiert. Dabei wird die in Android native Listener Funktionalität verwendet, wobei der Presenter als `OnClickListener` dient. Er ist sich über die Instanz der Container Klasse bewusst und interagiert direkt mit dessen Attributen und Methoden.

Das Model besteht aus verschiedenen Klassen, welche die Kacheltypen aus den Vorgaben widerspiegeln. Da diese dezentral vorliegen und es gemäß der geplanten MVP-Implementierung hier zu einer sehr komplexen Struktur von Events kommen würde, wurde sich dazu entschieden, die Kommunikation von Model zu Presenter über Rückgabewerte einzelner Methoden zu gestalten. Die entgegengerichtete Kommunikation verläuft nach Entwurf, also über direkte Aufrufe von Methoden und Attributen. Die aktuellen Operanden im Stack und die zuvor eingetragenen Operanden in der Historie werden jeweils als Listen im Presenter geführt.

Um mit dem Presenter jegliche Eingabe verarbeiten zu können, wird diesem die Kachelidentität im Event übergeben. Diese wird ihm übergeben und er führt je nach Typ verschiedene Operationen durch.

- Bei Identifikation einer Kachel vom Typ Operand, wird dieser entweder an die aktuelle Eingabe zur Anlage von neuen Operanden angehängt, z.B. 2 und 3 zu 23, oder in den Stack und in die Historie geschoben.
- Bei einer Operation wird diese mit so vielen Operanden wie möglich ausgeführt und das Ergebnis in Stack und Historie geschoben.
- Die Verarbeitung von Einstellung ist sehr individuell und nur gegebenenfalls werden Inhalte in Stack und Historie geschoben. Ein Beispiel für eine Einstellung ist die Taste `AC` (All Clear) im Taschenrechner. Diese entfernt jegliche Eingaben aus

dem aktuellen Stack.

- Identifiziert der Presenter eine Kachel vom Typ Stack oder Historie, so wird diese wie ein Operand behandelt.

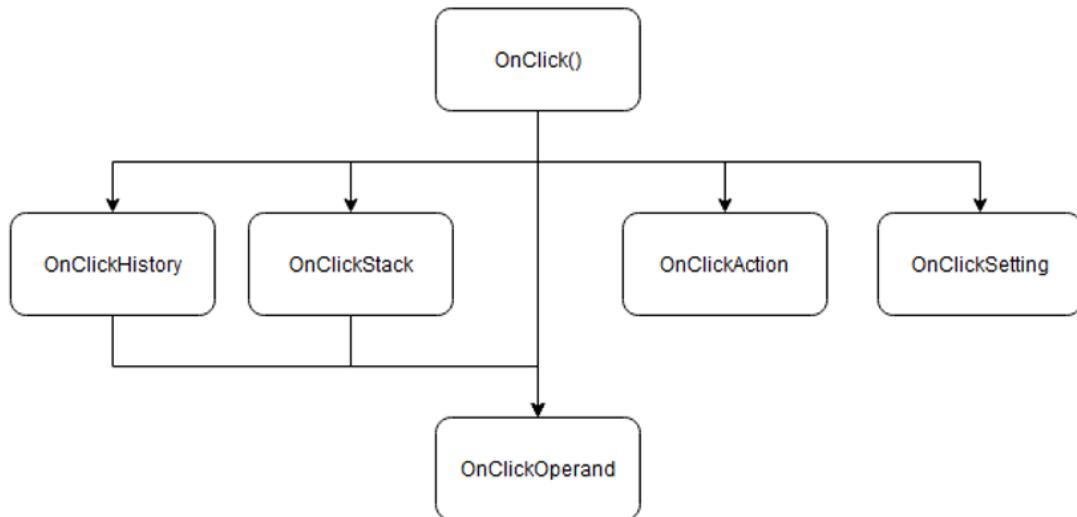


Abbildung 21: Click Event Handling vom Presenter²⁶

²⁶eigene Darstellung

4.4.5 Planung der View [Tom]

Für die Planung der View entschieden wir zuerst, dass die Applikation beim Start eine Kachelgestaltung in die Main Activity lädt. Dadurch erhält man eine Gridstruktur, in welcher die Kacheln unabhängig voneinander liegen. Diese Kachel geben ihren Inhalt an den Presenter weiter.

Dadurch, dass die Layouts angepasst werden können, muss das Laden dynamisch implementiert werden. Die Kacheln werden durch das Layout als Buttons mit Text generiert und setzt dann die Listener aus dem Observer Pattern. Es gibt einen Kachelcontainer, welcher eine Liste aller Kacheln hat und eine Liste beider benötigten Stacks. Der Kachelcontainer soll mit Hilfe des Factory Patterns erstellt werden. Dabei wird ein String aus dem Speicher geladen und in den Container umgewandelt. Die Factory kann auch aus einem Layout den dazu passenden String erstellen und mit einer Bezeichnung versehen.

Da die Kachel vom Kontext abhängig sind können diese nicht im Voraus geladen werden und es können keine Layouts gleichzeitig geladen werden.

Es gibt ein Schema für Kacheln, damit diese nicht vom Kontext abhängig sind. Die Factory erstellt aus den Schemata konkrete Kacheln mit Typ und Inhalt. Der Inhalt muss an den Typ angepasst werden und damit die Kacheln nicht überladen werden können, werden Untertypen erstellt, die mit Hilfe eines Mappings ausgewählt werden können. Es werden erst beim Laden eines Layout die Schemata in tatsächliche Kacheln übertragen. Änderungen der Kacheln laufen auch exklusiv über die Schemata.

XXX ganz viele punkte

4.4.6 Planung der Menüsteuerung [Istogu]

Nachdem beschrieben wurde, wie die Planung für die Umsetzung der Anforderung erfolgte ("alle Kacheln sind gleichbedeutend, das heißt keine Kachel besitzt einen unveränderlichen Sonderstatus"²⁷) beschäftigt sich das folgende Unterkapitel mit der Anforderung, dass jeder Kachel jede Rolle zugewiesen werden kann.

In der Planungsphase wurde sich stärker an den Anforderung orientiert, zumal sie mögliche Implementierung skizzieren. Laut der Anforderung existiert ein Kacheltyp `CalculatorFunction`, der unter anderem "auch die Zuordnung einer Funktionalität zu einer Kachel" beinhaltet. Daher war die Idee, dass die Applikation zwei Activities beinhalten soll. Einmal eine Activity, die die Nutzung des Taschenrechners ermöglicht, und eine andere Activity, die einen "Bearbeiten-Modus" darstellen soll. Von dort aus könnte der Anwender sein Layout anpassen. Jedoch wurde diese Idee verworfen, weil sie auch mit den Anforderung teilweise kollidiert, dass die Funktionalitäten über eine minimale Menge von Interaktion erreicht werden soll. Stattdessen wurde entschieden den Ansatz zu verfolgen, der in der XXX Abbildung 7 zu sehen ist. Dieses Menü soll aufgerufen werden, wenn der Anwender eine Kachel länger gedrückt hält. Dabei soll ein Menü geöffnet werden, dass eine Auswahl an unterstützten Kacheltypen anbietet. Abhängig davon was ausgewählt wird, erscheint ein weiteres Untermenü. In der nachfolgende Abbildung wird dies anhand der Zuweisung eines Bruches verdeutlicht. Die letzten Untermenüs dabei sind den Operanden vorbehalten.

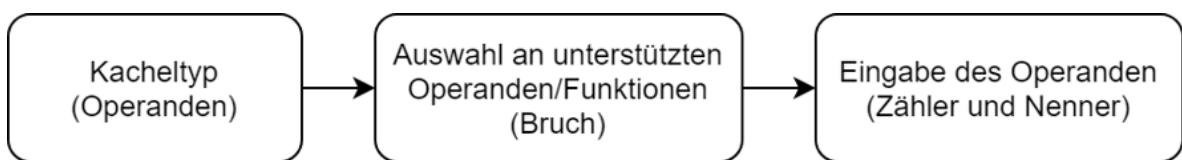


Abbildung 22: Ablaufdiagramm für die Zuweisung eines Bruches an einer Kachel²⁸

XXX

²⁷vgl. Seifert, T. (2020)

²⁸eigene Darstellung

4.5 Geplante Aufgabenverteilung im Team

Name	Aufgaben
Tim Schwenke	Einrichtung, Verwaltung und Bereitstellung der Versionsverwaltung Architekturentwurf der Kalkulationsorchestrierung LaTeX-Beauftragter Projekttagebuch führen Dokumentation anfertigen
Hendrik Falk	Gruppenweite Koordination des Projektes Erstellung der simplen Berechnungen und Sondertasten Architekturentwurf der Front-End- und Back-End-Kommunikation Projekttagebuch führen Dokumentation anfertigen
Tim Meinerzhagen	Planung und Implementierung der Java-Schnittstelle Auswahl der zu verwendenden Math Bibliothek Projekttagebuch führen Dokumentation anfertigen
Getuart Istogu	Spezifikation und Dokumentation des Datenmodells und der Datenschnittstellen Projekttagebuch führen Dokumentation anfertigen
Dennis Gentges	Planung der Navigation zwischen den Activities und die einen Activity-Wechsel auslösenden Ereignisse Projektplanung Konzeptionierung der Benutzeroberfläche Paper Prototype Projekttagebuch führen

	Dokumentation anfertigen
Tom Bockhorn	Planung der Navigation zwischen den Activities und die einen Activity-Wechsel auslösenden Ereignisse Konzeptionierung der Benutzeroberfläche Projekttagebuch führen Dokumentation anfertigen
Khang Pham	Verantwortlichkeit für Erscheinungsbild der Arbeit, Vollständigkeit gemäß Gliederungsstruktur, Formatierung des Quellcodes im Anhang. Konzeptionierung der Benutzeroberfläche Projekttagebuch führen Dokumentation anfertigen
Jannis Luca Keienburg	LaTeX-Beauftragter Projekttagebuch führen Dokumentation anfertigen

5 Beschreibung des Projektverlaufs

5.1 Tatsächliche Aufgabenverteilung im Team

XXX

5.2 Teammeeting-Protokolle

Datum	Dauer	Beschreibung
03.09.2019	200	<p>Gruppenmeeting 1 - Projektauswahl</p> <p><i>Teilnehmer: Bockhorn, Falk, Gentges, Istogu, Keienburg, Meinerzhagen, Pham, Schwenke</i></p> <p>Auswahl des Projekttyps. Entscheidung für die Entwicklung einer Android-App.</p> <p>Erste Einarbeitung in die Thematik. Lesen des bereitgestellten Dokuments mit Aufgabenstellung, groben Anforderungen und organisatorisches.</p> <p>Konzepterarbeitung auf Papier. Vorstellung und Diskussion verschiedener Ansätze.</p>
04.09.2019	110	<p>Gruppenmeeting 2 - Grobes Konzept</p> <p><i>Teilnehmer: Bockhorn, Falk, Gentges, Istogu, Keienburg, Meinerzhagen, Pham, Schwenke, Herr Prof. Dr. Seifert</i></p> <p>Einigung mit dem Dozenten auf einen Ansatz für den Taschenrechner.</p> <p>Absprachen über Meeting-Protokolle, Studienarbeit und Projekttagebücher</p> <p>Ausarbeitung des Konzepts für den Taschenrechner. Hier wurden dem Auftraggeber Herr Seifert mehrere Konzepte vorgestellt und gemeinsam mit ihm genaue Anforderungen erarbeitet.</p> <p>Aufteilung der Gruppe in das GUI-, Architektur- und Backend Team</p>
05.09.2019	60	<p>GUI-Team-Meeting 1 - Prototyp</p> <p><i>Teilnehmer: Bockhorn, Gentges, Pham</i></p> <p>Erstellung des Mid-Fidelity Prototypen in Power Point.</p>
	60	<p>Architektur-Team-Meeting 1 - Konzept</p> <p><i>Teilnehmer: Falk, Meinerzhagen, Schwenke</i></p>

		Grobe Konzeptionierung der Architektur des Projektes inklusive grobem Aufbau des Projektes und der Schnittstellen
	60	<p>Backend-Team-Meeting 1 - Funktionen sammeln</p> <p><i>Teilnehmer: Istogu, Keienburg</i></p> <p>Brainstorming über geplante Operatoren, Operanden und Einstellungen</p>
	60	<p>Gruppenmeeting 3 - Abstimmung der Teams</p> <p><i>Teilnehmer: Bockhorn, Falk, Gentges, Istogu, Keienburg, Meinerzhagen, Pham, Schwenke</i></p> <p>Besprechung des aktuellen Stands der Architektur, der Backend-Ideen und des Prototypen.</p> <p>Diskussion über Umsetzung und Workflow der App.</p> <ul style="list-style-type: none"> – Wie sollen die einzelnen Kacheln funktionieren? – Wie sollen die Kacheln miteinander interagieren? – Wie könnte die Architektur der App aussehen? <p>Absprache der Projektplanungsergebnisse</p>
13.09.2019	120	<p>GUI-Team-Meeting 2 - Mid-Fidelity Prototyp</p> <p><i>Teilnehmer: Bockhorn, Gentges, Pham</i></p> <p>Erweiterung des Mid-Fidelity Prototyps in PowerPoint inklusive Workflow mit Links</p> <p>Finalisierung der geplanten Funktionalitäten des Prototypen</p>
17.09.2019	60	<p>Architektur-Team-Meeting 2 - Klassendiagramm</p> <p><i>Teilnehmer: Falk, Istogu, Meinerzhagen, Schwenke</i></p> <p>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.</p>

		<p>Welche Library soll für Mathe-Funktionalitäten benutzt werden? JScience und die bereits mitgelieferte Standardbibliothek.</p> <p>Darstellungsformen der Elemente mit ASCI-Art oder LaTeX</p>
	90	<p>GUI-Team-Meeting 3 - Workflow</p> <p><i>Teilnehmer: Bockhorn, Gentges, Pham</i></p> <p>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.</p> <p>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.</p> <p>Wie soll die Eingabe von Funktionen im Graph Operand funktionieren? Nur möglich mit bereits vorhandenen Elementen in der Oberfläche. Es öffnet sich keine Tastatur.</p>
23.09.2019	120	<p>GUI-Team-Meeting 4 - Prototyp</p> <p><i>Teilnehmer: Bockhorn, Gentges, Pham</i></p> <p>Finalisierung des Mid-Fidelity Prototyps inklusive der Funktionalitäten</p> <p>Durchplanung des Workflows</p>
26.09.2019	150	<p>Gruppenmeeting 4 - Abstimmung der Teams</p> <p><i>Teilnehmer: Bockhorn, Falk, Gentges, Istogu, Keienburg, Meinerzhagen, Pham, Schwenke</i></p> <p>Vorstellung des Mid-Fidelity Prototyps und Einholung von Feedback</p> <p>Erklärung der geplanten Systemarchitektur</p> <p>Darstellung der geplanten Programmodule</p>

09.10.2019	90	<p>Gruppenmeeting 5 - Abstimmung der Teams</p> <p><i>Teilnehmer: Bockhorn, Falk, Gentges, Istogu, Keienburg, Meinerzhagen, Pham, Schwenke</i></p> <p>Vorstellung des Backend-Entwurfs für Teammitglieder, die für das Frontend zuständig sind.</p> <p>Vorstellung des Frontend-Entwurfs für Teammitglieder, die für das Backend zuständig sind.</p> <p>Diskussion über Verbindung von Frontend und Backend. Wie abgekoppelt lässt sich der Calculator wirklich realisieren?</p> <p>Vorstellung der Hauptbibliothek die für die (aufwändigen) Rechnungen wie Nullstellenberechnung benutzt werden soll.</p> <p>Warum Apache Commons Math und nicht JScience?</p> <p>Diskussion des Programm-Workflows.</p>
05.01.2020	120	<p>Gruppenmeeting 6 - Abstimmung der Teams</p> <p><i>Teilnehmer: Bockhorn, Falk, Gentges, Istogu, Keienburg, Meinerzhagen, Pham, Schwenke</i></p> <p>Aufnahme des aktuellen Projektstands.</p> <p>Besprechung des weiteren Vorgehens.</p> <p>Abstimmung der Aufgaben.</p> <p>Besprechung des geplanten Frontends.</p> <p>Besprechung / Lösung von Problemen.</p>
	180	<p>GUI-Team-Meeting 5 - Abstimmung der Teams</p> <p><i>Teilnehmer: Bockhorn, Gentges, Pham</i></p> <p>Besprechung des Frontends und Einarbeitung des Feedbacks</p> <p>Recherche nach Emulator Skins</p> <p>Erste Drafts des Layouts</p> <p>Anlegen der Kacheln (Style resource)</p>
10.01.2020	90	<p>GUI-Team-Meeting 6 - Layouts</p> <p><i>Teilnehmer: Bockhorn, Gentges, Pham</i></p>

		Weitere Standardlayouts finalisiert Kachelarchitektur für Frontend finalisiert
14.01.2020	90	<p>Gruppenmeeting 7 - Integration der Komponenten</p> <p><i>Teilnehmer: Bockhorn, Falk, Gentges, Istogu, Keienburg, Meinerzhagen, Pham, Schwenke</i></p> <p>Zusammenführung Frontend Backend</p> <p>Präsentation des Frontends durch das GUI-Team.</p> <p>Besprechen von MVP-Umsetzung in Android.</p> <p>Backend Unit-Testing Fortschritte.</p> <p>Serialisierung der Stacks zur Session-Sicherung.</p>
24.01.2020	240	<p>Architektur- und Backend-Team-Meeting</p> <p><i>Teilnehmer: Falk, Istogu, Keienburg, Meinerzhagen, Schwenke</i></p> <p>Detaillierte Ausarbeitung der Architektur im Backend.</p> <p>Programmieren im Team inklusive der Zusammenführung der Features</p> <p>Umbau der Programmstruktur zur Anpassung an MVP</p>
28.01.2020	90	<p>Prototyp Vorstellung</p> <p><i>Teilnehmer: Bockhorn, Falk, Istogu, Meinerzhagen, Schwenke Herr Prof. Dr. Seifert</i></p> <p>Gespräch mit Herr Prof. Dr. Thomas Seifert über den aktuellen Stand des Projekts und im Anschluss daran eine Nachbesprechung innerhalb des Teams.</p>

	<p>Vorstellung:</p> <ul style="list-style-type: none"> – 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). <p>Ergebnis:</p> <ul style="list-style-type: none"> – 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.
90	<p>Gruppenmeeting 8 - Nachbesprechung Vorstellung</p> <p><i>Teilnehmer: Bockhorn, Falk, Gentges, Istogu, Keienburg, Meinerzhagen, Pham, Schwenke</i></p> <p>Ergebnisse der Nachbesprechung</p> <ul style="list-style-type: none"> – "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.
120	<p>GUI-Team-Meeting 7 - Tests & Menüs</p> <p><i>Teilnehmer: Bockhorn, Gentges, Pham</i></p>

		Funktionstests des Frontends Beginn der Implementierung der Menüführung
03.02.2020	90	<p>Gruppenmeeting 9 - Absprachen & Dokumentation</p> <p><i>Teilnehmer: Bockhorn, Falk, Gentges, Istogu, Keienburg, Meinerzhagen, Pham, Schwenke</i></p> <p>Besprechen des aktuellen Stands der App.</p> <p>Was muss noch unbedingt umgesetzt werden?</p> <p>Aufteilung der noch offenen Kapitel in der Ausarbeitung.</p> <p>Neues Kapitel "Einleitung" mit Motivation.</p> <p>Anpassung einiger Kapitelbezeichnungen an Gegebenheiten des Projekts.</p> <p>Koordination der Ausarbeitung.</p>
	50	<p>Backend-Team-Meeting 2 - Finalisierung</p> <p><i>Teilnehmer: Istogu, Keienburg</i></p> <p>Absprache der letzten zu anzupassenden Funktionalitäten</p>
05.02.2020	90	<p>Gruppenmeeting 10 - Finalisierung & Dokumentation</p> <p><i>Teilnehmer: Bockhorn, Falk, Gentges, Istogu, Keienburg, Meinerzhagen, Pham, Schwenke</i></p> <p>Ergebnisbesprechung für das App-Debugging.</p> <p>Gemeinsames Arbeiten an der Dokumentation.</p>
<i>Summe der Dauer aller Gruppenmeetings: 1.180 Minuten</i>		
<i>Summe der Dauer aller GUI-Team-Meetings: 1.020 Minuten</i>		
<i>Summe der Dauer aller Architektur- Team-Meetings: 360 Minuten</i>		
<i>Summe der Dauer aller GUI-Team-Meetings: 110 Minuten</i>		

5.3 Projekttagebücher

5.3.1 Tom Bockhorn

Datum	Dauer	Beschreibung
03.09.2019	200	Gruppenmeeting 1
	120	Recherche zur programmatischen Umsetzung von Taschenrechnern
	30	Einrichten von Android Studio mit Emulator
04.09.2019	110	Gruppenmeeting 2
	90	Auf Basis des Feedbacks wurde das Konzept-Design überarbeitet und angepasst
05.09.2019	60	GUI-Team-Meeting 1
	60	Gruppenmeeting 3
	80	Konzeptionierung der Bearbeitungsfunktionalitäten des Taschenrechners. Erstellung und Bearbeitung des Mid-Fidelity-Prototypens Besprechung des aktuellen Standes
13.09.2019	120	GUI-Team-Meeting 2
17.09.2019	90	GUI-Team-Meeting 3
23.09.2019	120	GUI-Team-Meeting 4
26.09.2019	150	Gruppenmeeting 4
09.10.2019	90	Gruppenmeeting 5
05.01.2020	120	Gruppenmeeting 6
	180	GUI-Team-Meeting 5
06.01.2020	90	Anlesen von Activity Verhalten in Android
	360	Konzeption der Architektur für generische Kacheln im Frontend Konzeption eines Containers für Kacheln im Frontend
07.01.2020	180	Aufteilung der Kacheln in kontextnah und kontextfern

10.01.2020	120	Entwicklung der Container mit aufgeteilter Kachelarchitektur Finalisierung der Container und Testen
	90	GUI-Team-Meeting 6
11.01.2020	90	Fehlerbehebung in den Containern Schulung anderer Teammitglieder über die generische Kachelarchitektur
14.01.2020	90	Gruppenmeeting 7
	120	Testen der Container Implementierung und Optimierung der Lade und Speichermodule
28.01.2020	90	Prototyp Vorstellung
	90	Gruppenmeeting 8
	120	GUI-Team-Meeting 7
02.02.2020	120	Einlesen in mögliche Menüführungen in Android
03.02.2020	120	Konzeption und Entwicklung der Rangauswahlmenüs für Stack und Historie
	60	Integration der Rangauswahlmenü in die restlichen Menüs
	90	Gruppenmeeting 9
04.02.2020	300	Fehler der Layout Container beheben Unterstützende Fehlerbehebende Arbeit in anderen Bereichen der Software
05.02.2020	90	Gruppenmeeting 10
<i>Summe der Dauer aller Aktivitäten: 3.840 Minuten</i>		

5.3.2 Hendrik Falk

Datum	Dauer	Beschreibung
03.09.2019	200	Gruppenmeeting 1
	x	x
	x	x
04.09.2019	x	x
	x	x
05.09.2019	x	x
	x	x
17.09.2019	x	x
	x	x
26.09.2019	x	x
	x	x
09.10.2019	x	x
	x	x
20.12.2019	x	x
	x	x
02.01.2020	x	x
	x	x
03.01.2020	x	x
	x	x
05.01.2020	x	x
	x	x
06.01.2020	x	x
	x	x
07.01.2020	x	x
	x	x
08.01.2020	x	x

	x	x
14.01.2020	x	x
	x	x
24.01.2020	x	x
	x	x
27.01.2020	x	x
	x	x
28.01.2020	x	x
	x	x
30.01.2020	x	x
	x	x
03.02.2020	x	x
	x	x
04.02.2020	x	x
	x	x
05.02.2020	x	x
	x	x
<i>Summe der Dauer aller Aktivitäten: x Minuten</i>		

5.3.3 Dennis Gentges

Datum	Dauer	Beschreibung
03.09.2019	200	Gruppenmeeting 1
	x	x
	x	x
04.09.2019	x	x
	x	x
05.09.2019	x	x
	x	x
08.09.2019	x	x
	x	x
10.09.2019	x	x
	x	x
13.10.2019	x	x
	x	x
17.09.2019	x	x
	x	x
23.09.2019	x	x
	x	x
26.09.2019	x	x
	x	x
01.10.2019	x	x
	x	x
09.10.2019	x	x
	x	x
16.11.2019	x	x
	x	x
05.01.2020	x	x

	x	x
12.01.2020	x	x
	x	x
14.01.2020	x	x
	x	x
24.01.2020	x	x
	x	x
28.01.2020	x	x
	x	x
31.01.2020	x	x
	x	x
02.02.2020	x	x
	x	x
03.02.2020	x	x
	x	x
05.02.2020	x	x
	x	x
<i>Summe der Dauer aller Aktivitäten: x Minuten</i>		

5.3.4 Getuart Istogu

Datum	Dauer	Beschreibung
03.09.2019	200	Gruppenmeeting 1
04.09.2019	110	Gruppenmeeting 2
04.09.2019	40	Ideen für mögliche Operatoren und Operanden sammeln
04.09.2019	10	Offizielle und interne Deadlines in Planner eintragen
05.09.2019	60	Gruppenmeeting 3
05.09.2019	60	Backend-Team-Meeting 1
17.09.2019	60	Architektur-Team-Meeting 2
26.09.2019	150	Gruppenmeeting 4
09.10.2019	90	Gruppenmeeting 5
12.10.2019	30	Ermitteln der benötigten Operanden-Klassen (z.B. Matrix etc.) Ermitteln der benötigten Operatoren-Klassen (Addition, Multiplikation usw.)
26.10.2019	30	Recherche auf JScience und Apache Commons Math Vergleichen der Ergebnisse in der Gruppe Bereitstellung der Klassen/des Quellcodes Überlegungen wie diese zu importieren sind
09.11.2019	80	Durchführung des Cognitive-Walkthroughs
04.01.2020	120	Gruppenmeeting 6
04.01.2020	120	Auseinandersetzung des bereits implementierten Code von den anderen Teammitglieder Implementierung des Backends (Potenz, Wurzel, Modulo) Auseinandersetzung über mögliche Gestaltung der Exception (+ Diskussion mit ein paar Teammitglieder)
14.01.2020	90	Gruppenmeeting 7
19.01.2020	200	Klasse: Integral implementieren und zugehörige UnitTest-Klasse geschrieben

		Bietet Apache Commons Math Integralrechnung an? Umsetzung der Funktion für das Integral Eigene Implementation für das Bilden von Stammfunktionen Testklasse geschrieben
19.01.2020	90	Erstellung von Testklassen für die Klassen Potenz, Wurzel, Modulo
22.01.2020	350	Klasse: Limes und zugehörige UnitTest-Klasse geschrieben
24.01.2020	240	Architektur- und Backend-Team-Meeting
28.01.2020	400	Klasse: UtilMatrix und zugehörige UnitTest-Klasse implementiert
28.01.2020	90	Gruppenmeeting 8
29.01.2020	120	Verständnis für die Frontend-Klassen und deren Interaktionen gewinnen
29.01.2020	150	Überlegung der Architektur für die Menügestaltung
29.01.2020	270	Vorherige Implementierung abstrahieren Implementierung der abstrakten Klasse DialogMenu
30.01.2020	300	Implementierung der Klasse InputTileType und Einbinden mit ClickListener
03.02.2020	90	Gruppenmeeting 9
04.02.2020	420	Dokumentation des Projektes
05.02.2020	90	Gruppenmeeting 10
05.02.2020	30	Projekttagbuch pflegen
05.02.2020	280	Dokumentation des Projekts (Teil 2)
<i>Summe der Dauer aller Aktivitäten: 4.370 Minuten</i>		

5.3.5 Jannis Keienburg

Datum	Dauer	Beschreibung
03.09.2019	200	Gruppenmeeting 1
	x	x
	x	x
04.09.2019	x	x
	x	x
05.09.2019	x	x
	x	x
17.09.2019	x	x
	x	x
09.10.2019	x	x
	x	x
12.10.2019	x	x
	x	x
26.10.2019	x	x
	x	x
01.01.2020	x	x
	x	x
02.01.2020	x	x
	x	x
03.01.2020	x	x
	x	x
04.01.2020	x	x
	x	x
11.01.2020	x	x
	x	x
14.01.2020	x	x

	x	x
15.01.2020	x	x
	x	x
16.01.2020	x	x
	x	x
17.01.2020	x	x
	x	x
22.01.2020	x	x
	x	x
23.01.2020	x	x
	x	x
28.01.2020	x	x
	x	x
31.01.2020	x	x
	x	x
01.02.2020	x	x
	x	x
02.02.2020	x	x
	x	x
03.02.2020	x	x
	x	x
04.02.2020	x	x
	x	x
05.02.2020	x	x
	x	x
<i>Summe der Dauer aller Aktivitäten: x Minuten</i>		

5.3.6 Tim Jonas Meinerzhagen

Datum	Dauer	Beschreibung
03.09.2019	200	Gruppenmeeting 1
	x	x
	x	x
04.09.2019	x	x
	x	x
05.09.2019	x	x
	x	x
	x	x
	x	x
	x	x
17.09.2019	x	x
	x	x
24.09.2019	x	x
26.09.2019	x	x
	x	x
09.10.2019	x	x
	x	x
05.11.2019	x	x
	x	x
05.01.2019	x	x
	x	x
07.01.2019	x	x
	x	x
12.11.2019	x	x
	x	x
13.11.2019	x	x

	x	x
08.01.2020	x	x
	x	x
14.01.2020	x	x
	x	x
24.01.2020	x	x
	x	x
26.01.2020	x	x
	x	x
27.01.2020	x	x
	x	x
28.01.2020	x	x
	x	x
02.02.2020	x	x
	x	x
04.02.2020	x	x
	x	x
05.02.2020	x	x
	x	x
<i>Summe der Dauer aller Aktivitäten: x Minuten</i>		

5.3.7 Khang Pham

Datum	Dauer	Beschreibung
03.09.2019	200	Gruppenmeeting 1
	x	x
	x	x
04.09.2019	x	x
	x	x
05.09.2019	x	x
	x	x
13.09.2019	x	x
	x	x
17.09.2019	x	x
	x	x
23.09.2019	x	x
	x	x
26.09.2019	x	x
	x	x
09.10.2019	x	x
	x	x
05.01.2020	x	x
	x	x
07.01.2020	x	x
	x	x
10.01.2020	x	x
	x	x
14.01.2020	x	x
	x	x
08.01.2020	x	x

Beschreibung des Projektverlaufs

	x	x
25.01.2020	x	x
	x	x
28.01.2020	x	x
	x	x
01.02.2020	x	x
	x	x
02.02.2020	x	x
	x	x
28.01.2020	x	x
	x	x
02.02.2020	x	x
	x	x
03.02.2020	x	x
	x	x
05.02.2020	x	x
	x	x
<i>Summe der Dauer aller Aktivitäten: x Minuten</i>		

5.3.8 Tim Schwenke

Datum	Dauer	Beschreibung
03.09.2019	200	Gruppenmeeting 1
	120	Recherche zur programmatischen Umsetzung von Taschenrechnern
	30	Einrichten von Android Studio mit Emulator
	30	Einrichten von zwei Git-Repositories für das Softwareprojekt selbst und die Dokumentation in Latex
	30	Erarbeitung Vorgehen Projekttagebücher
04.09.2019	110	Gruppenmeeting 2
05.09.2019	60	Architektur-Team-Meeting 1
	60	Gruppenmeeting 3
	30	Finalisierung und Abgabe Projekttagebuch
14.09.2019	180	Erstellung mehrerer Konzepte für die Kalkulationsorchestrierung
15.09.2019	180	Einlesen in generische Programmierung mit Java
16.09.2019	100	Suchen und Ausprobieren von Mathebibliotheken für Java
	60	Genauerer Vergleich mithilfe von Beispielen zwischen Apache Commons Math und JScience durchführen und Entscheidung getroffen.
17.09.2019	60	Architektur-Team-Meeting 2
21.09.2019	180	Erstellung des Operanden Konzeptes
12.11.2019	120	Implementierung der Klasse Operand
	60	Implementierung von Hilfsklassen (DoubleComparator etc.)
26.09.2019	150	Gruppenmeeting 4
05.10.2019	360	Prototypische Entwicklung zweier Konzepte für Orchestrierung
06.10.2019	360	Umsetzung der finalen Schnittstelle für generische Kalkulationen

09.10.2019	90	Gruppenmeeting 5
	90	Projektdokumentation
10.10.2019	60	Optimierung des Operand-Stacks
05.11.2019	30	Finalisierung und Abgabe Projekttagebuch
05.01.2020	120	Gruppenmeeting 6
08.01.2020	30	Finalisierung und Abgabe Projekttagebuch
14.01.2020	90	Gruppenmeeting 7
16.01.2020	120	Optimierung verschiedener Actions (inc. Debugging)
	60	Schreiben von Unit-Tests für einige Actions
24.01.2020	240	Architektur- und Backend-Meeting
28.01.2020	90	Prototyp Vorstellung
	90	Gruppenmeeting 8
01.02.2020	120	Projektdokumentation
03.02.2020	90	Gruppenmeeting 9
	50	Backend-Team-Meeting 2
05.02.2020	90	Gruppenmeeting 10
	30	Finalisierung und Abgabe Projekttagebuch
	360	Projektdokumentation
<i>Summe der Dauer aller Aktivitäten: 4.330 Minuten</i>		

5.4 Beschreibung von Problemen

Bei der Erstellung der Applikation kam es in den jeweiligen Teams zu verschiedenen Arten von Problemen, welche in den nachfolgenden Punkten näher erläutert werden.

5.4.1 Programmierkenntnisse [Gentges]

Aufgrund mangelnder vorhergegangener Erfahrungen in der Programmierung sowie mit GitHub und Latex fiel es mir zu Beginn schwer konkrete Programmieraufgaben zu übernehmen. Dadurch benötigte ich eine wesentlich größere Einarbeitungszeit als meine Kollegen was zu einem gewissen zeitlichen Verzug im Front End Team führte.

Nach der Einarbeitung versuchte ich mich an ersten Codierungsaufgaben, was jedoch insbesondere am Anfang der zusätzlichen Kontrolle meiner Kollegen bedurfte. Leider kam es hierbei bei der Codeimplementierung meinerseits zu Fehlern weshalb der Code nicht kompiliert werden konnte und zusätzlicher Aufwand bei der Fehlerfindung und Erklärung angefallen ist.

5.4.2 Vergleich von Dezimalzahlen

XXX

5.4.3 Implementierung generischer Kacheln

XXX

5.4.4 Teamkommunikation in unterschiedlichen Umgebungen [Falk]

Ein großes Problem für die Kommunikation innerhalb des Teams war die Situation, dass wir an verschiedenen Standorten arbeiteten. Die meisten von arbeiteten im Chempark Leverkusen, jedoch hatten wir auch Teammitglieder in Wuppertal und Monheim. Das erschwerte die Kommunikation, da vieles digital verlagert werden musste. Dadurch viel die sehr wichtige Komponente des Face-to-Face-Meetings sehr oft raus, jedoch nicht komplett, da wir uns oft auch an den Samstagen austauschen konnten, an welchen wir an Vorlesungsveranstaltungen teilnahmen.

Bei der digitalen Kommunikation gab es allerdings auch verschiedene Probleme. Zum einen wurde einigen von uns keine Einrichtungen zum Telefonieren bereitgestellt, wo-

durch diese bei Meetings zwar zuhören konnten, aber selber nur über Text kommunizieren konnten. Auch waren die meisten von uns bei der Bayer AG angestellt, weshalb wir oft auch interne Kommunikationsmöglichkeiten nutzten, zu welchen die Teammitglieder, welche von der Currenta GmbH & Co. OHG beschäftigt wurden, nicht immer Zugriff hatten. Das lag daran, dass in der Bearbeitungszeit Bayer seine Anteile an Currenta verkaufte und daher die vorher eng verbundenen IT-Systeme getrennt werden mussten.

6 Dokumentation der Software

6.1 Dokumentation der Paketstruktur des Android-Projektes [Falk]

Die Applikation liegt im Paket `de.fhdw.wip.rpntilecalculator` und ist in einzelne Klassen unterteilt, welche in eine Ordnerstruktur eingebettet sind. Diese Ordnerstruktur orientiert sich an dem gewählten UI Design Pattern MVP. Dabei gibt es einen Model Ordner. Dieser ist in vier weitere Ordner unterteilt, welche den Kachelarten nachempfunden sind und einen für den Stack. In diesen befinden sich verschiedene Klassen, die zusammen das Model bilden. Im `Presenter`-Ordner befindet sich lediglich der Presenter. Der `View`-Ordner besteht wieder aus mehreren Teilen. Zum einen aus der `MainActivity`, welches die Hauptansicht der Applikation darstellt. Dazu befinden sich dort Typen und Grundklassen der Kacheln. Es gibt noch die drei Ordner `Layout`, `Menu` und `Schemes`. In `Layout` befinden sich Funktionen zum Darstellen, Speichern und Laden der Kacheln. In `Schemes` befinden sich Vorlagen für Tiles und in `Menu` sind sämtliche Menüs gespeichert.

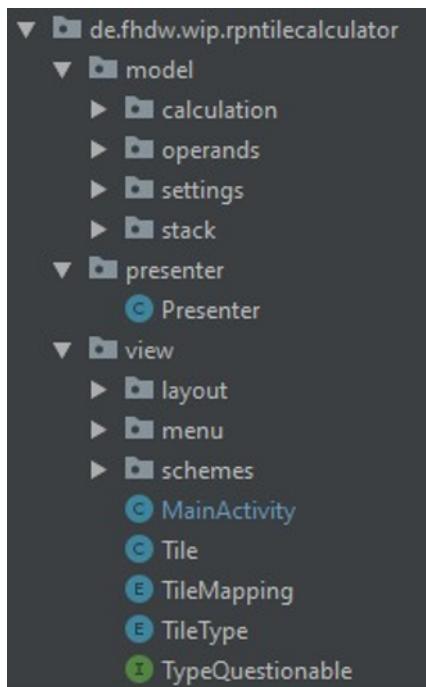


Abbildung 23: Ordnerstruktur²⁹

²⁹eigene Darstellung (Screenshot aus Android Studio)

³⁰eigene Darstellung

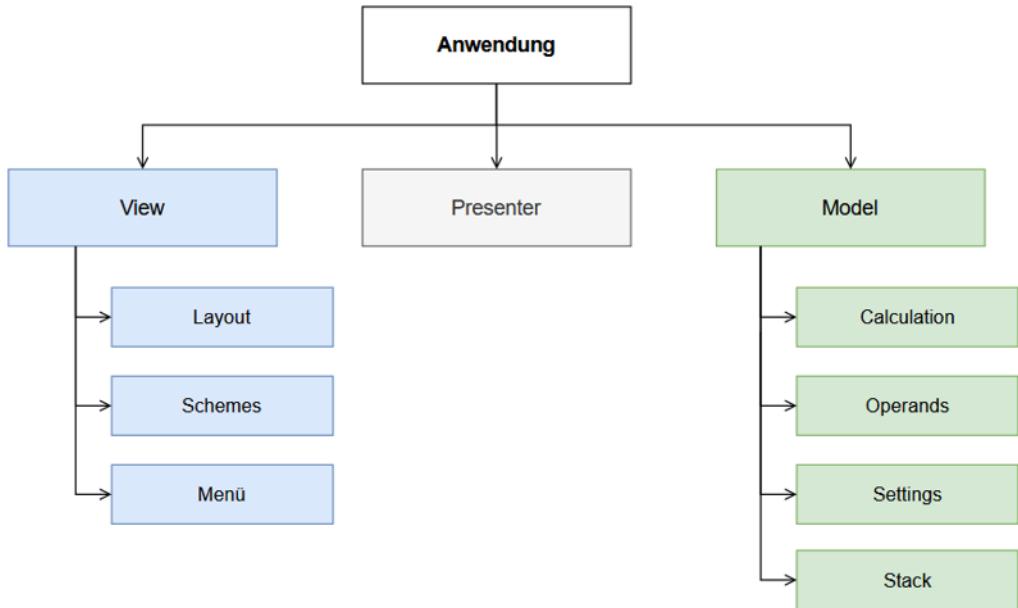


Abbildung 24: Ordnerstruktur Schema³⁰

6.2 Dokumentation der View

In den folgenden Kapiteln wird die Implementierung des Frontends der Applikation, der sogenannten View, und jeglichen zugehörigen Implementierungen, dargestellt.

6.2.1 Activities [Bockhorn]

Die Applikation wird durch den Aufruf der sogenannten `MainActivity` vom Android System beim Öffnen der Anwendung gestartet. Da die Menüführung nicht auf Activities basiert und jegliche Layouts dynamisch geladen werden, bleibt die `MainActivity` die einzige Activity der Applikation

Im MVP Pattern ist die `Activity` der View zugeordnet. Sie lädt unter anderem vertikale und horizontale Layouts vor und stellt eine Methode zur Darstellung dieser bereit. Auch wird ein Listener erstellt, der nach Änderungen der Bildschirmausrichtung horcht und passende Layouts lädt.

6.2.1.1 Klasse: `MainActivity`

Beschreibung: Activity, die beim Starten der Anwendung aufgerufen wird und das Layout `activity_main.xml` aufruft. Dieses Layout ist jedoch leer, da alle Inhalte dy-

namisch geladen werden. Es lädt auch Taschenrechner-Layouts vor und stellt Methoden zur Darstellung dieser bereit.

Methode `onCreate` lädt die Layouts vor, erstellt einen `Listener` ob die Orientierung sich verändert und öffnet das erste Layout.

Methode `setTileLayout` lädt ein Layout in die View.

6.2.2 Generische Kachelgestaltung [Tom]

Die generische Gestaltung der Kacheln wurde gemäß dem Entwurf so umgesetzt, dass eine Trennung zwischen Kacheln, die als Buttons vom Kontext der Applikation abhängig sind, und Kacheln, die vom Kontext unabhängig sind, ermöglicht wird. Dies dient unter anderem dem Vorladen von Layouts zur Optimierung der Ladeperformance. Außerdem wird so die Definition des Kacheltypen und dessen spezialisierte Inhalte auf die vom Kontext unabhängigen Kachelschemata (`TileSchemes`) ausgelagert.

Als logische Abtrennung kann man sich merken, dass Kacheln (`Tiles`) lediglich für die Kommunikation mit dem Nutzer zuständig sind, während die `TileSchemes` die restliche Frontend-Logik beinhalten.

6.2.3 Kontextbezogene Kacheln [Bockhorn]

Die Tiles werden mit Bezug auf den Kontext, also die aktuell dargestellte Activity, erstellt. Sie dienen als generische Kachelklasse, dessen Funktionalitäten unabhängig vom Kacheltypen gebraucht werden.

Ein Tile, als Unterklasse des `AppCompatButton` von Android, wirft beim Anklicken ein Event, welches von registrierten `OnClickListeners` abgefangen werden kann. Für die einzelnen Tiles wird der `Presenter` als `OnClickListener` registriert, sodass dieser die vom Nutzer getätigten Klicke verarbeiten kann. Zusätzlich wird dem `Tile` als `OnLongClickListener` das Menü `TileTypeInput` übergeben, welches bei langem Klicken geöffnet wird und somit die Bearbeitung des Kacheltypen ermöglicht.

Die Informationen über den Kacheltypen des `Tiles` befindet sich im `TileScheme` des `Tiles`. Werden nun die Inhalte, oder gar der Kacheltyp des `Tiles` verändert, so wird das Aussehen des `Tiles` anhand des aktuellen `TileSchemes` aktualisiert. Ähnlich

³¹eigene Darstellung

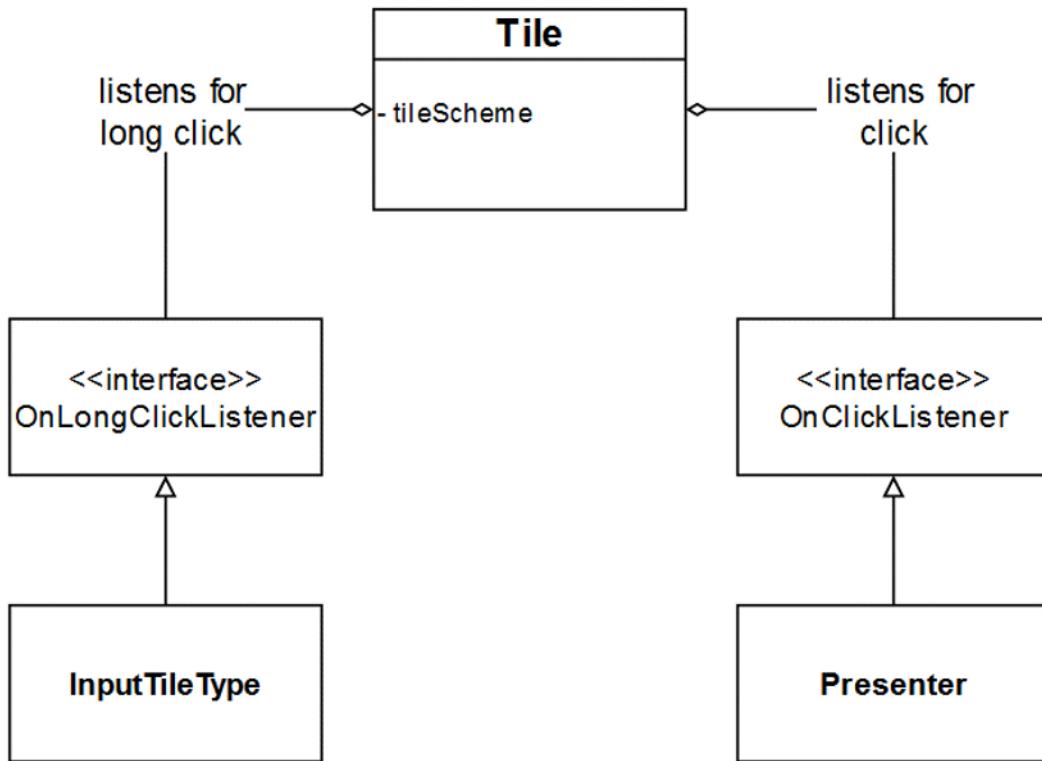


Abbildung 25: Listener von Tile³¹

stellt ein `Tile` verschiedene Animationen bereit, die zum Beispiel beim Laden oder Speichern des `Tiles` aufgerufen und abgespielt werden können.

6.2.3.1 Tile [Bockhorn]

Superklasse: `AppCompatButton`

Beschreibung: Agiert als Button auf einem Kontext, dessen Inhalt und Typ durch ein `TileScheme` definiert sind. Einfaches Klicken wird vom `Presenter` abgefangen und gedrückt halten öffnet ein Auswahlmenü für die Kachelart.

Methode `update` Aktualisiert das `Tile` mithilfe eines neuen `Schemes`. Setzt dabei Hintergrund Ressource und Text

Methode `enableMenulistener` Aktiviert die Menüfunktion für dieses `Tile`

6.2.4 Kontextfremde Kacheln [Gentges]

Die `TileSchemes` stehen in keiner Beziehung zum Kontext der Anwendung. Dennoch wird in ihnen der Text und das Design des letztendlichen `Tiles` gespeichert. Im Gegensatz zu den `Tiles` differenzieren `TileSchemes` hier zwischen Kachelarten. Die Grundeigenschaften eines `TileSchemes`, also die Information darüber um welche Art von Kachel es sich handelt, und was genau der Inhalt ist, werden in der Superklasse `TileScheme` definiert. Ersteres liegt in Form eines sogenannten `TileMappings` vor, welches in einem folgenden Kapitel näher erläutert wird XXX. Für jede der bestehenden Kachelarten wurde eine Unterklasse implementiert, welche die Eigenschaften von `TileScheme` erbt und weitere artbezogene Inhalte definiert. In Kombination mit einem `Tile` können so andere Komponenten der Applikation, wie beispielsweise der `Presenter`, die Kachelart des `TileScheme` erfragen und den vorliegenden Inhalt für die auszuführenden Prozesse extrahieren.

- Das `ActionTileScheme` ist die Implementierung eines `TileScheme` vom Typ Action, (auch `Operator`) der Inhalt und Typ um eine instanzierte Action erweitert. Dank der ähnlich polymorphen Struktur der `Action`s, kann so mit einem simplen Methodenaufruf jede Form von `Action`, die im Schema hinterlegt ist, angesprochen und ausgeführt werden.
- Ein `SettingTileScheme` erbt ebenfalls von `TileScheme`, aber liefert zusätzlich Informationen über ein sogenanntes `Setting`, welches ähnlich wie eine `Action` über eine vererbte Methode anzusprechen ist.
- Ein `OperandTileScheme` ist gleich aufgebaut, beinhaltet aber Operanden, die als Grundlage für die Kalkulation mit `Action`s dienen.
- Das Klicken auf ein `Tile` des Stacks wird intern gleich gehandhabt, wie das Klicken eines Operanden. Jedoch besitzen Tiles aus dem Stack eine Reihenfolge, die im `StackTileScheme` als Rang hinterlegt wird. Um die gleiche Behandlung eines potenziellen Operanden im Stack zu gewährleisten, erbt `StackTileScheme` nicht von `TileScheme`, sondern von `OperandTileScheme`.
- Ein `HistoryStackTile` ist ebenfalls so aufgebaut, wie ein `StackTileScheme`, und erbt daher dessen `Attribute`.

Die Erstellung einzelner `TileSchemes` erfolgt mithilfe des Factory Method Design-Pattern. Hierzu stellt die abstrakte Superklasse `TileScheme` zwei Methoden zur Verfü-

gung. Innerhalb dieser Methoden wird anhand des 'TileMappings' zwischen den verschiedenen Kachelarten differenziert und der passende Konstruktor der Unterklasse aufgerufen. In einer Methode erfolgt die Erstellung des TileScheme Inhalts durch die Einlese eines Zeichenkette. Dies findet beispielsweise bei der Kreierung von TileSchemes, nach der Auslese eines im internen Speicher gespeicherten Layouts, Anwendung. Als zweite Option kann aus einem bereits bestehenden Operanden ein TileScheme der Art Operand, Stack oder Historie erstellt werden. Dadurch, dass Stack und Historie eines Layouts regelmäßig aktualisiert werden müssen, bietet sich die performante Erstellung von TileSchemes mithilfe bestehender Operanden, gegenüber der aufwendigen erneuten Auslese aus einem String, an.

Im Kontrast zu den Tiles, die eine Methode zum Aktualisieren der Inhalte bereitstellen, müssen TileSchemes mithilfe der genannten Methoden rekreiert werden. Kann eine Kachelart nicht identifiziert werden, oder sind die Angaben über den Inhalt der Kachel unkorrekt, so wird ein ErrorTileScheme erstellt, das visuell von jeder anderen Art zu unterscheiden ist und den Entwicklern somit über den Fehler Bescheid gibt.

6.2.4.1 Abstrakte Klasse: TileScheme

Beschreibung: Abstrakte Superklasse, die neben den artübergreifenden Inhalten der TileSchemes, Methoden zur Kreierung dieser gemäß des Factory Method Design-Pattern bereitstellt.

Methode `createTileScheme` liefert in Abhängigkeit vom angegebenen TileMapping eine Instanz einer Unterklasse des TileSchemes. Die Erstellung von TileSchemes für Actions und Settings ist lediglich durch die Eingabe von Zeichenketten möglich. TileSchemes der Arten Operanden, Stack und Historie können zusätzlich aus bestehenden Operanden erstellt werden.

6.2.4.2 Klasse: ActionTileScheme

Superklasse: TileScheme

Beschreibung: TileScheme für Operatoren. Action, die ausgeführt werden soll, als Zusatzinformation. Inhalt ist der Text der Kachel.

6.2.4.3 Klasse: SettingTileScheme

Superklasse: TileScheme

Beschreibung: `TileScheme` für Einstellungen. `Settings`, die ausgeführt werden soll, als Zusatzinformation. Inhalt ist der Text der Kachel.

6.2.4.4 Klasse: `OperandTileScheme`

Superklasse: `TileScheme`

Beschreibung: `TileScheme` für Operanden. Bei der Erstellung aus Text wird der String mithilfe Reflections (XXX) zu einem Operanden übersetzt.

6.2.4.5 Klasse: `StackTileScheme`

Superklasse: `OperandTileScheme`

Beschreibung: `TileScheme` für Operanden im Stack. Wird genauso behandelt wie ein `OperandTileScheme`, doch verfügt zusätzlich über eine Information über den Rang der Kachel im gesamten Stack.

6.2.4.6 Klasse: `StackTileScheme`

Superklasse: `OperandTileScheme`

Beschreibung: `TileScheme` für Operanden im Stack. Wird genauso behandelt wie ein `OperandTileScheme`, doch verfügt zusätzlich über eine Information über den Rang der Kachel im gesamten Stack.

6.2.4.7 Klasse: `HistoryTileScheme`

Superklasse: `StackTileScheme`

Beschreibung: `TileScheme` für Operanden in der Historie. Wird genauso behandelt wie ein `OperandTileScheme`, doch verfügt zusätzlich über eine Information über den Rang der Kachel im Gesamtstack.

6.2.4.8 Klasse: `ErrorTileScheme`

Superklasse: `TileScheme`

Beschreibung: `TileScheme` für die Ausnahmesituation, dass ein reguläres `TileScheme` nicht geladen werden konnte

6.2.5 Kacheltypdefinition durch Enumartion [Pham]

Wie zuvor erläutert, erstellt die Klasse `TileSchemes` ein passendes Schema anhand eines sogenannten `TileMappings`. Ein `TileMapping` ist im Endeffekt eine konkrete Definition einer Kachel im Frontend-System. Hierzu beinhaltet das entsprechende `TileMapping` jegliche Informationen, die für eine bestimmte Kachel von Relevanz ist.

Durch die Deklaration als Enum werden diese Informationen mit einer Zeichenkette zur Einlese in Verbindung gebracht.

Für den Operator `-` (`Minus`) ist dies beispielsweise die Kachelart (`Action / Operator`), eine Referenz auf die passende `Action` Klasse und den Anzeigetext der Kachel `-`, die mit dem Text `A_MINUS` betitelt werden.

Die einzelnen Definitionen liegen in Form einer Java Enumeration vor, da diese Ein Beispiel für eine solche Definition sieht wie folgt aus:

```
A_MINUS(TileType.ACTION, Minus.getInstance(), "")
```

Neben Operatoren werden im Enum die anderen Datentypen ebenfalls angelegt. Technisch wurde dies durch die Verwendung von überladenen Konstruktoren ermöglicht und die `TileSchemes` wissen welche Informationen gebraucht werden. Zur Definition der Kachelart selbst wurde eine weitere Enumeration mit folgenden Werten angelegt:

- `Stack`
- `History`
- `Operand`
- `Action`
- `Setting`
- `Error`

Zu den Kachelarten sind Design Ressourcen hinterlegt, welche die Gestaltung der Kacheln je nach Typ definiert.

6.2.5.1 Enumeration: TileMapping

Beschreibung: Definiert Kacheln und dessen wichtigste Elemente. Wird außerdem zur Einlese von Texten genutzt.

Bsp: `A_MINUS(TileType.Action, Minus.getInstance(), " -")`

6.2.5.2 Enumeration: TileType

Beschreibung: Definiert die verschiedenen Kachelarten und ihre Design Ressourcen

Bsp: `ACTION(R.drawable.tile_operator_blue)`

6.2.6 Gruppierung der Kacheln im Layout Container [Bockhorn]

XXX

6.2.7 Implementierung der Menüsteuerung [Istogu]

In diesem Abschnitt wird die Umsetzung der Planung für die Menüsteuerung aufgegriffen. Als erstes wird die Architektur des Quellcodes näher dargelegt. Darauf anschließend wird sich mit der Implementierung der Zwischenmenüs und der Eingabeansicht für eine Auswahl an Operanden befasst.

6.2.7.1 Klasse: Dialogmenu [Istogu]

Bei der `DialogMenu` handelt es sich um eine abstrakte Klasse, das von `View.OnClickListener` implementiert. Der Konstruktor erwartet folgende Parameter: `MainActivity context, Tile, displayTile, DialogMenu, last`.

In dem Konstruktor wird ein `Dialog`-Objekt erstellt. Dabei werden initial bestimmte Attribute für das `Dialog` festgelegt, die für alle nachfolgenden Menüs gelten soll. Als Beispiel kann hier aufgeführt werden, dass der Titel festgelegt wird. Zudem wird bestimmt, dass das Fenster zentral erscheint. Dieser Dialog wird erst angezeigt, wenn der `OnClickListener` ein Click vom vorherigen Dialog registriert. Zusätzlich wird `ContentViewID` über die Methode `setContentView()` festgelegt, die auf die einzelne `xml`-Dateien referenzieren.

Die Methode `dismissAll()` sorgt dafür, dass beim Verlassen des Dialoges alle vorherigen Dialoge mitgeschlossen werden. Die Umsetzung des Schließens der vorherigen Dialogen erfolgt das rekursive Aufrufen der Methode `dismissAll()` von dem höheren gestellten Dialog-Objekt. Daher wird eine Referenz zu dem letzten Dialog benötigt. Der erste Dialog erhält ein `null`-Objekt, das bei der `dismissAll` überprüft wird.

Grundsätzlich werden Dialoge geschlossen, wenn der Nutzer außerhalb des Dialoges mit der Benutzeroberfläche reagiert oder über den Quellcode `dialog.dismiss()` aufgerufen wird.

6.2.7.2 Klasse: InputTileType [Istogu]

Die Klasse erbt von der abstrakten Klasse `DialogMenu` und verweist über den `contentView` auf die `.xml`-Datei. Dabei werden die Layouts (`.xml`) mithilfe der Klasse `R.java` verbunden. Diese Klasse wird während des Kompilierens Referenzen zu den Ressourcen der Applikation erstellt.

Zu den Ressourcen zählen unter anderem Bilder, `XML`-Ansichten und deren Komponenten wie Buttons. Die Referenzen zu den Buttons werden danach verwendet, um auf die Objekte über die Methode `dialog.findViewById()` zuzugreifen. In dem Konstruktor werden zu den jeweiligen Buttons ein `OnClickListener` festgesetzt, der ein neues Objekt vom Typ `InputTileMapping` erzeugt.

6.2.7.3 Klasse: InputTileMapping [Istogu]

XXX

6.2.7.4 Klasse: InputMenuFactory [Bockhorn]

XXX

6.2.7.5 Klasse: InputDouble [Gentges]

XXX

6.2.7.6 Klasse: InputFraction [Gentges]

XXX

6.2.7.7 Klasse: InputPolynomial [Gentges]

XXX

6.2.7.8 Klasse: 6.2.5.8 Darstellung der Beziehungen von den Menüsteuerungsklassen [Gentges]

XXX

6.3 Dokumentation der Models

6.3.1 Operanden [Schwenke]

Für ein gut funktionierendes Backend ist ein einheitliches Datenmodell essenziell. So gibt es in der Kernbibliothek von Java einige Klassen für die Repräsentation von mathematischen Bausteinen, wie z.B. dem Bruch. Das gleiche gilt für *Apache Commons Math*.

Jedoch sind die meisten dieser Klassen sehr spezialisiert und miteinander nicht kompatibel. Deswegen wurde innerhalb des Projektteams entschieden für jeden unterstützten Operanden eine eigene Klasse zu entwickeln, die jedoch alle von der gleichen abstrakten Klasse `Operand` erben sollen.

Dadurch wird sichergestellt, dass es eine einheitliche Schnittstelle gibt, über die alle Operanden angesprochen werden können. Beschreiben kann man diese neuen Klassen auch als *Wrapper*, da der Großteil der eigentlichen Funktionalitäten Teil der darunterliegenden Klassen ist.

So soll z.B. `OMatrix` ein Wrapper um die Klasse `Array2DRowRealMatrix` sein. Damit kann man einerseits das einfache und einheitliche Interface nutzen und falls notwendig direkt auf das darunterliegende Objekt zugreifen, um spezielle Operationen durchführen zu können.

Ein wichtiger Bestandteil dieser Wrapper sind auch die unterschiedlichen Konstruktoren. So kann jeder Operand aus einem einzelnen String konstruiert werden.

Im Folgenden wird die gemeinsame Schnittstelle dargestellt. Gemeinsam haben alle diese Methoden, dass sie ohne Seiteneffekte auskommen. Statt das vorliegende Objekt zu verändern wird ein neues Objekt erstellt und zurückgegeben. Dies macht das Umkehren von Methodenaufrufen einfach. Man muss lediglich das zurückgegebene Objekt entfernen und das alte weiternutzen.

6.3.1.1 Klasse: Operand [Schwenke]

`Operand turnAroundSign()` : Dreht alle Vorzeichen im Operand um. Erreicht wird das grundsätzlich durch Multiplikation mit `-1`. Konkreter muss dafür z.B. bei einer Matrix eine Multiplikation mit einem Skalar durchgeführt werden.

`Operand negateValue()` : Negiert den Operanden unabhängig von den Vorzeichen.

Bei einer Matrix werden also alle Elemente negativ.

`Operand inverseValue()` : Die Inverse (wenn vorhanden) des jeweiligen Operanden wird berechnet.

`Boolean equalsValue(Operand o)` : Vergleicht den aktuellen Operanden mit einem übergebenen Operanden.

`String toString()` : Wandelt das Objekt in eine String-Repräsentation um.

`<T extends Object> get()` : Gibt das darunterliegende Objekt zurück. Im Falle von `OMatrix` ist dies eine `Array2DRowRealMatrix`. Bei einem Tupel ein `Array` von `Doubles`.

Auffallend ist hier der geringe Umfang der Schnittstelle der Operanden, da diese doch eigentlich die Grundlage für einen Taschenrechner bilden sollten. So kann man eine Matrix multiplizieren, addieren, dividieren und das alles in unterschiedlichsten Kombinationen mit anderen Arten von Operanden. Tatsächlich sind die meisten Kalkulationen – wie in der Projektplanung entworfen – in eigenen Klassen angelegt. Somit besteht eine Trennung zwischen Datenhaltung (den Operanden) und den Operationen, die auf den Daten ausgeführt werden können (Actions).

6.3.1.2 Klasse: ODouble [Meinerzhagen]

Wrapper für ein Double

Eine Dezimalzahl, welche intern einen `double` zur Datenspeicherung verwendet.

6.3.1.3 Klasse: OFraction [Meinerzhagen]

Wrapper für Common Math Fraction

Implementierung eines Bruchs. Die Zähler und Nenner sind durch `Doubles` umgesetzt. Im Gegensatz zu den meisten anderen Operanden gibt es hier mehr als zwei Konstruktoren. So kann man einmal Zähler und Nenner als zwei ganze Zahlen übergeben. Wird ein einzelnes `Double` übergeben, wird daraus automatisch ein Bruch erstellt. Alternativ kann direkt eine `Fraction` übergeben werden.

6.3.1.4 Klasse: OSet [Meinerzhagen]

Wrapper für ein Double - Set

Implementierung einer Menge. Das gleiche Element kann nicht mehrmals enthalten sein. Die Reihenfolge spielt keine Rolle. Umgesetzt ist die Klasse mit einem angepassten `Set`. Erstellen kann man ein `OSet` aus einem `Array` von `Doubles` oder einem String.

6.3.1.5 Klasse: OTuple [Meinerzhagen]

Wrapper für ein Double - Array

Implementierung eines Tupels. Das gleiche Element kann mehrmals enthalten sein. Die Reihenfolge spielt eine Rolle. Erstellen kann man ein `OTuple` aus einer übergebenen Liste oder einem String.

6.3.1.6 Klasse: OMatrix [Meinerzhagen]

Wrapper für eine Apache Commons Array2DRowRealMatrix

Implementierung einer Matrix. Das darunterliegende Objekt ist eine `RealMatrix`, die wiederum eine `Array2DRowRealMatrix` enthält. Die Matrix an sich ist in einem zweidimensionalen Array von `Doubles` umgesetzt. Erstellen kann man eine `OMatrix` aus einem String oder einem zweidimensionalen `Array` von `Doubles`.

6.3.1.7 Klasse: OPolynom [Meinerzhagen]

Wrapper für eine Apache Commons PolynomialFunction

Implementierung eines Polynoms. Grundsätzlich entspricht das Polynom einer einer Sequenz von Dezimalzahlen. Abhängig von der Position in der Sequenz entscheidet sich, was der Exponent der jeweiligen Dezimalzahl ist. Somit müssen Funktionen immer in dieser Normalform vorliegen. Erstellen kann man ein `OPolynom` aus einem String oder einem `Array` von `Doubles`.

6.3.1.8 Klasse: OEmpty [Meinerzhagen]

Leerer Wrapper

Ein leerer Operand. Da alle `Operand`-Klassen Wrapper sind, ist technisch auch ein leerer Wrapper notwendig. Die hier implementierten Methoden aus der Schnittstelle stellen lediglich *Stubs* dar und haben keine Funktion.

6.3.1.9 Klasse: DoubleFormatter [Meinerzhagen]

Zentrale Anlaufstelle um `Doubles` in einen formattierten String umzuwandeln. Die einzige hier enthaltene Methode ist

```
String format(double d)
```

6.3.1.10 Klasse: DoubleComparator [Meinerzhagen]

Dezimalzahlen sind in den meisten Sprachen in dem IEE 754 Format implementiert. Das hat zur Folge, dass Konversionen und Änderungen einer konkreten Dezimalzahl zu Rundungsfehlern führen. Das ist auch in Java der Fall. Möchte man nun die beiden Dezimalzahlen `1.1` und `1.1` mit der Standardmethode in Java vergleichen, wird man in fast allen Fällen ein `false` zurückbekommen. Das liegt an den zuvor angesprochenen Rundungsfehlern.

Da innerhalb dieses Projekts alles auf `Doubles` basiert ist es notwendig für dieses Problem eine Lösung zu finden. Dafür wurde diese Klasse entwickelt. Alle Vergleiche müssen über diese Klasse erfolgen.

`Boolean isEqual(double d1, double d2)` vergleicht zwei `Doubles` miteinander. Dies erfolgt über ein Delta. Zunächst wird die Differenz zwischen beiden Zahlen berechnet und anschließend überprüft, ob die Differenz unter einer definierten Grenze liegt. Alle weiteren Methoden in dieser Klasse benutzen diese Methode.

Weitere Methoden vergleichen `Arrays` und `Sets` miteinander.

6.3.2 Operationen

6.3.2.1 Klasse: Plus [Falk]

Eingabeparameter: Zwei Werte, der erste Wert ist der erste Summand, der zweite der zweite Summand. Die erlaubten Kombinationen sind:

- `ODouble` und `ODouble`
- `ODouble` und `OFraction`
- `OFraction` und `ODouble`
- `ODouble` und `OSet`
- `OSet` und `ODouble`

- `ODouble` und `OMatrix`
- `OMatrix` und `ODouble`
- `ODouble` und `OPolynom`
- `OPolynom` und `ODouble`
- `ODouble` und `OTupel`
- `OTupel` und `ODouble`
- `OFraction` und `OFraction`
- `OFraction` und `OSet`
- `OSet` und `OFraction`
- `OFraction` und `OMatrix`
- `OMatrix` und `OFraction`
- `OFraction` und `OPolynom`
- `OPolynom` und `OFraction`
- `OFraction` und `OTuple`
- `OTupel` und `OFraction`
- `OMatrix` und `OMatrix`
- `OPolynom` und `OPolynom`
- `OTuple` und `OTuple`

Rückgabewerte: Der berechnete Wert mit dem Datentyp des ersten Summanden.

Beschreibung: Die Klasse verfügt über 23 öffentlich ansprechbare Methoden. Je nach übergebenen Parametern wird bestimmt, welche öffentliche Methode gemeint ist. Anschließend wird die Berechnung durchgeführt.

6.3.2.2 Klasse: Minus [Falk]

Eingabeparameter: Zwei Werte, der erste Wert ist der Minuend, der zweite der Subtrahend. Die erlaubten Kombinationen sind:

- `ODouble` und `ODouble`

- ODouble und OFraction
- OFraction und OFraction
- OFraction und ODouble
- OSet und ODouble
- OSet und OFraction
- OMatrix und OMatrix
- OMatrix und ODouble
- OMatrix und OFraction
- OPolynom und OPolynom
- OPolynom und ODouble
- OPolynom und OFraction
- OTuple und OTuple
- OTuple und ODouble
- OTuple und OFraction

Rückgabewerte: Der berechnete Wert mit dem Datentyp des ersten, übergebenen Wertes.

Beschreibung: Die Klasse verfügt über 15 öffentlich ansprechbare Methoden. Je nach übergebenen Parametern wird bestimmt, welche öffentliche Methode gemeint ist. Anschließend wird die Berechnung durchgeführt.

6.3.2.3 Klasse: Minus [Falk]

Eingabeparameter: Zwei Werte, der erste Wert ist der Dividend, der zweite der Divisor.
Die erlaubten Kombinationen sind:

- ODouble und ODouble
- ODouble und OFraction
- OFraction und OFraction
- OFraction und ODouble

- OSet und ODouble
- OSet und OFraction
- OMatrix und ODouble
- OMatrix und OFraction
- OPolynom und OPolynom
- OPolynom und ODouble
- OPolynom und OFraction
- OTuple und OTuple
- OTuple und ODouble
- OTuple und OFraction

Rückgabewerte: Der berechnete Wert mit dem Datentyp des ersten, übergebenen Wertes.

Beschreibung: Die Klasse verfügt über 14 öffentlich ansprechbare Methoden. Je nach übergebenen Parametern wird bestimmt, welche öffentliche Methode gemeint ist. Bevor die Berechnung durchgeführt wird, wird bestimmt ob der Divisor null ist. Wenn ja wird eine Fehlermeldung ausgegeben. Andernfalls wird die Berechnung durchgeführt.

6.3.2.4 Klasse: Times [Falk]

Eingabeparameter: Zwei Werte, der erste Wert ist der Multiplikator, der zweite Wert der Multiplikand. Die erlaubten Kombinationen sind:

- ODouble und ODouble
- ODouble und OFraction
- OFraction und ODouble
- ODouble und OSet
- OSet und ODouble
- ODouble und OMatrix
- OMatrix und ODouble

- ODouble und OPolynom
- OPolynom und ODouble
- ODouble und OTupel,
- OTupel und ODouble
- OFraction und OFraction
- OFraction und OSet
- OFraction und OMatrix
- OMatrix und OFraction
- OFraction und OPolynom
- OFraction und OTupel
- OMatrix und OMatrix
- OPolynom und OPolynom
- OTupel und OTupel

Rückgabewerte: Der berechnete Wert.

Beschreibung: Die Klasse verfügt über 20 öffentlich ansprechbare Methoden. Je nach übergebenen Parametern wird bestimmt, welche öffentliche Methode gemeint ist. Anschließend wird die Berechnung durchgeführt.

6.3.2.5 Klasse: Root [Falk]

Eingabeparameter: Zwei Werte, der erste Wert ist der Radikand, der zweite Wert der Wurzelexponent. Die erlaubten Kombinationen sind:

- ODouble und ODouble
- ODouble und OFraction
- OFraction und ODouble
- OFraction und OFraction
- OMatrix und ODouble
- OMatrix und OFraction

Rückgabewerte: Der berechnete Wert, Datentyp ist der des Radikanden.

Beschreibung: Die Klasse verfügt über 6 öffentliche Methoden. Anhand der übergebenen Parameter wird bestimmt, welche öffentliche Methode aufgerufen wird. In der Methode wird die Wurzel berechnet und zurückgegeben.

6.3.2.6 Klasse: Modulo [Falk]

Eingabeparameter: Zwei Werte des Typs `ODouble`, der erste Wert ist der Dividend, der zweite der Divisor.

Rückgabewerte: Der berechnete Wert als `ODouble`.

Beschreibung: Die Klasse verfügt über eine öffentliche Methode. In der Methode wird der Modulo-Wert der beiden übergebenen Parameter berechnet.

6.3.2.7 Klasse: Zeros [Keienburg]

Eingabeparameter: Eine Funktion des Typs `OPolynomial`

Rückgabewerte: Ein Set des Typs `OSet` mit den berechneten Nullstellen. Die Nullstellen werden als `Set` zurückgegeben, da so verhindert wird das für Funktionen wie $x^2+0*x+0$ zweimal die gleiche Nullstelle zurückgegeben wird.

Einstieg in die Klasse: Öffentliche Methode, die die übergebenen Parameter entgegennimmt. Die weiteren Methoden für die Berechnung werden anschließend von der Öffentlichen Methode aus aufgerufen. Methoden der Klasse `Zeros`:

`calculateZeros` : Die Methode erhält eine Funktion vom Typ `OPolynomial`. Anschließend wird bestimmt, von welchem Grad die übergebene Funktion ist. Wenn die Funktion ersten Grades ist, wird die Methode `zerosTypeOne` aufgerufen, ansonsten die Methode `zerosTypeTwo`. Die beiden Methoden geben die gefundenen Nullstellen als Array zurück, das Array wird anschließend von `calculateZeros` zurückgegeben.

`normalOrQuadraticFunction` : Bestimmt, ob eine, als `Double` Array übergebene Funktion, ersten oder zweiten Grades ist. Dabei wird die Länge des Arrays getestet. Ist die Länge des Arrays 3, so wird die Funktion als Quadratische Funktion bestimmt. Wenn die Länge 2 ist, als einfache Funktion.

`zerosTypeOne` : Die Methode berechnet die Nullstellen für Funktionen ersten Grades. Die Funktion wird als Double Array übergeben. Die berechnete Nullstelle wird als

Double Array zurückgegeben.

zerosTypeTwo : Die Methode berechnet die Nullstellen für Funktionen zweiten Grades. Die Funktion wird als Double Array übergeben. Mithilfe der Mitternachtsformel wird die erste und die zweite Nullstelle bestimmt. Die gefundenen Nullstellen werden als Double Array zurückgegeben.

Anmerkung: Wenn eine übergebene Funktion keine Nullstellen besitzt, der Rückgabewert in Java also `NaN` ist, wird eine Fehlermeldung ausgegeben, dass eine Nullstellenberechnung nicht möglich ist.

Einschränkungen: Die Nullstellenberechnung ist nur für Funktionen ersten und zweiten Grades möglich.

Unit-Tests für die Klasse:

1. Eingabe Funktion: `2x + 4`. Erwartetes Ergebnis: `(-2)`
2. Eingabe Funktion: `2x^2+4x+0`. Erwartetes Ergebnis: `(-2, 0)`
3. Eingabe Funktion: `x^2+4x-4`. Erwartetes Ergebnis: `(-4,828, 0,828)`

Die Tests waren erfolgreich.

6.3.2.8 Klasse: HighAndLowPoints [Keienburg]

Eingabeparameter: Eine Funktion des Typs `OPolynomial`

Rückgabewerte: Ein Tupel des Typs `OTuple` mit den berechneten Extremwerten. Die Werte an den geraden Positionen innerhalb des Tupels sind die X-Werte, der jeweils folgende Wert ist der zugehörige Y-Wert. Bsp.: Tupel [0] = X-Wert des ersten Extremwertes, Tupel [1] = Y-Wert des ersten Extremwertes, Tupel [2] = X-Wert des zweiten Extremwertes, Tupel [3] = Y-Wert des zweiten Extremwertes.

Beschreibung: Die Klasse erhält eine Funktion vom Typ `ODouble`. Zuerst wird die Ableitung der Funktion bestimmt, anschließend die Nullstellen der Ableitung. Für die erhaltenen Nullstellen wird der zugehörige Y-Wert berechnet. Die berechneten Werte werden als `OTuple` zurückgegeben.

Einstieg in die Klasse: Öffentliche Methode, die die übergebenen Parameter entgegennimmt. Die weiteren Methoden für die Berechnung werden anschließend von der Öffentlichen Methode aus aufgerufen.

Methoden der Klasse `HighAndLowPoints`:

`getHighAndLowPoints` : Einstiegspunkt der Klasse, ruft die Methode `calculateHighAndLowPoints` für die weitere Berechnung der Extremwerte auf. Gibt sie als Double Array zurück.

`getFunctionAsDouble` : Wandelt eine übergebene Funktion vom Typ `OPolynom` in ein Double Array um.

`calculateHighAndLowPoints` : Berechnet die Nullstellen einer übergebenen Funktion vom Typ `OPolynom`. Berechnet zuerst die Ableitung der Funktion, anschließend die Nullstellen der Ableitung. Die Werte der Nullstellen werden anschließend in die ursprüngliche Funktion eingesetzt und so die Y-Werte berechnet. Die berechneten Nullstellen werden als Double Array zurückgegeben.

Einschränkungen : Die Nullstellenberechnung ist nur für Funktionen zweiten Grades möglich. Demnach ist eine Berechnung der Hoch- und Tiefpunkte nur für Funktionen dritten oder zweiten Grades möglich.

Unit-Tests für die Klasse:

- Eingabe Funktion: $2x^2+4x+6$. Erwartetes Ergebnis: (-1|4)
- Eingabe Funktion: $3x^2+6x-4$. Erwartetes Ergebnis: (-1|-7)
- Eingabe Funktion: $-2*x^2+4x+12$. Erwartetes Ergebnis: (1|14)

Die Tests waren erfolgreich.

6.3.2.9 Logarithm [Keienburg]

Eingabeparameter: Ein Wert des Typs `ODouble` oder zwei Werte vom Typ `ODouble` (erster ist die Basis)

Rückgabewerte: Ein Wert des Typs `ODouble`

Beschreibung: Die Klasse berechnet den Logarithmus. Die Art des Logarithmus ist abhängig von den übergebenen Parametern. Wird nur ein Wert übergeben, wird der natürliche Logarithmus der Zahl berechnet. Wenn zwei Parameter übergeben werden, ist der erste Parameter die Basis des Logarithmus, der zweite der Wert, zu für den der Logarithmus berechnet werden soll.

Der Einstieg in die Klasse erfolgt über zwei öffentliche Methoden. Die Methode wird anhand der übergebenen Parameter (ein oder zwei Werte vom Typ `ODouble`) ausge-

wählt. In der Öffentlichen Methode findet die Berechnung des Logarithmus (Natürlicher oder Logarithmus zu einer bestimmten Basis) statt. Wenn ein Wert kleiner gleich null eingegeben wird, wird eine Fehlermeldung ausgegeben.

Unit-Tests für die Klasse: 4 Unit-Test für die Berechnung der des natürlichen Logarithmus (ersten 4), 3 Unit-Tests für die Berechnung des Logarithmus für eine beliebige Basis.

- Eingabe 10. Erwartetes Ergebnis: 2.302585092994046
- Eingabe: 5. Erwartetes Ergebnis: 1.6094379124341003
- Eingabe: 97. Erwartetes Ergebnis: 4.574710978503383
- Eingabe: e. Erwartetes Ergebnis: 1
- Eingabe: Basis 5, Wert 10. Erwartetes Ergebnis: 1.4306765580733933
- Eingabe: Basis 10, Wert 10. Erwartetes Ergebnis: 1
- Eingabe: Basis e, Wert 10. Erwartetes Ergebnis: 2.302585092994046

Die Tests waren erfolgreich.

6.3.2.10 Logarithm10 [Keienburg]

Eingabeparameter: Ein Wert des Typs ODouble

Ausgabeparameter: Ein Wert des Typs ODouble

Beschreibung: Die Klasse berechnet den Logarithmus zur Basis 10. Wenn ein Wert kleiner gleich null eingegeben wird, wird eine Fehlermeldung ausgegeben.

Unit-Tests für die Klasse:

- Eingabe 10. Erwartetes Ergebnis: 1
- Eingabe: 5. Erwartetes Ergebnis: 0.6989700043360189
- Eingabe: 97. Erwartetes Ergebnis: 1.9867717342662448

Die Tests waren erfolgreich.

6.3.2.11 Derivation [Keienburg]

Eingabeparameter: Eine Funktion des Typs OPolynom

Rückgabewert: Eine Funktion des Typs OPolynom

Beschreibung: Die Klasse berechnet die Ableitung zu einer übergebenen Funktion des Typs OPolynom und gibt sie als Funktion vom Typ OPolynom zurück.

Einstieg in die Klasse: Öffentliche Methode, die die übergebenen Parameter entgegennimmt. Die weiteren Methoden für die Berechnung werden anschließend von der Öffentlichen Methode aus aufgerufen.

Methoden der Klasse Derivation:

getFunctionAsDouble: Wandelt eine gegebene Funktion vom Typ OPolynom in ein Double Array um.

derivate: Berechnet die Ableitung einer gegebenen Funktion. Die Funktion wird zuerst in ein Double Array umgewandelt, anschließend abgeleitet und als Funktion vom Typ OPolynom zurückgegeben.

Unit-Tests für die Klasse:

- Eingabe $2x^2+4x+6$. Erwartetes Ergebnis: $4x+4$
- Eingabe: $7x+9$. Erwartetes Ergebnis: 7
- Eingabe: $12x^2-3+12$. Erwartetes Ergebnis: $24x-1$
- Eingabe: $1.5x^2+2x+7$. Erwartetes Ergebnis: $2.25x+2$

Die Tests waren erfolgreich.

6.3.2.12 Integral [Istogu]

Diese Klasse erbt von der Klasse „Action“. Daher sind die Methoden „with()“ bereits implementiert. Diese Klasse berechnet entweder die Stammfunktion von dem übergebene Polynomfunktion oder das Integral der Funktion.

Für die Bildung der Stammfunktion benötigt die Methode „getAntiderivative()“ eine „OPolynom“ und gibt die Stammfunktion als Typ „OPolynom“ zurück. Beim „Aufleiten“ entsteht eine Konstante, die jede Wert annehmen kann. In der App wird festgelegt, dass diese Konstante immer 0 ist. Zweck hinter der Definition ist, dass mit dem OPolynom weitergerechnet werden kann, da diese nicht mit weiteren Variablen arbeiten kann.

Für das Berechnen des Integrals getSimpsonIntegrator() werden folgende Parametertypen erwartet: OPolynom oPolynom, double lowerBound, double upperBound. Das

Integral für den vorgegebenen Bereich wird als ODouble zurückgegeben. Diese Methode realisiert die Simpsonregel für die Integration von realen Funktionen, die in dem genutzten Mathebibliothek als Klasse SimpsonIntegrator implementiert wurde.

Nach der Implementierung der Klasse erstellte ich dazu gehörig eine Unit-Test-Klasse. Im Nachfolgenden werden die Ergebnisse dargestellt, die alle erfolgreich waren.

`getSimpsonsIntegrator()` 1. Eingabe: OPolynom: $x^2 - 8x + 17$, untere Grenze: 2, obere Grenze 5

Erwartetes Ergebnis: 6

`getAntiderivative()` 1. Eingabe: OPolynom: $-3,21x^3 + 3x + 7$ Erwartetes Ergebnis: OPolynom: $-0,8025x^4 + 1,5x^2 + 7x + c$

Bestimmung des Grenzwertes einer Funktion [Istogu]

Für die Umsetzung wurde die Funktion “value(double x)” der Klasse “PolynomialFunction” verwendet, die aus der Mathebibliothek “Apache Commons Math” hervorgeht. Zudem bietet die Double-Klasse die Repräsentierung von positiv und negativ Unendlich an, die auch für die Methoden verwendet wurde.

Für die Bestimmung des Grenzwertes wurde sich an der Methode der numerischen Annäherung orientiert. Hierbei wird der Grenzwert durch die Annäherung von links und rechts des untersuchten Punktes bestimmt. Daher wurde die Anforderung in drei Methoden untergliedert, einmal die Methode “limitFromBelow(...)” und “limitFromAbove(...)”, die jeweils die Annäherung von verschiedenen Richtung bestimmt. Die Rückgabewerte der beiden Methoden werden in der Methode “limit(...)” miteinander verglichen. Zweck der Vergleichsabfrage ist es Definitionslücken abzufangen. Als Beispiel wird folgende Funktion angenommen:

$$f(x) = 1/x$$

Wenn für die Funktion sowohl der linksseitige als auch der rechtsseitige Grenzwert bestimmt wird, folgt folgendes:

³²eigene Darstellung

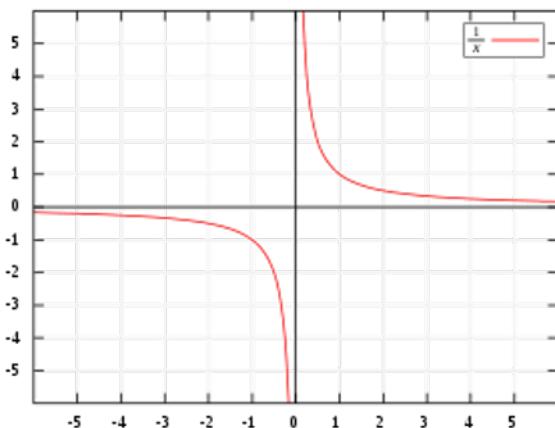


Abbildung 26: Grafische Darstellung der Funktion³²

6.4 Überblick über die Activities der App bzw. der Funktionen

6.5 Dokumentation der Navigation zwischen Activities

6.6 Dokumentation der Activity-übergreifenden, persistenten Datenhaltung

6.7 Dokumentation der programmatischen Beiträge der Teammitglieder

6.7.1 Tom Bockhorn

6.7.2 Hendrik Falk

6.7.3 Dennis Gentges

6.7.4 Getuart Istogu

6.7.5 Jannis Keienburg

6.7.6 Tim Jonas Meinerzhagen

6.7.7 Khang Pham

6.7.8 Tim Schwenke

6.7.8.1 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 1: 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 2: 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 2 beginnt mit der Extraktion der Klasse jedes übergebenen Operanden. 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 Fehlerarten in `CalculationException` zusammengefasst und weitergegeben. Ansonsten wird das Ergebnis zurückgegeben.

6.8 Beschreibung von Problemen

6.8.1 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 Entwickler 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.

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:

³³Atlassian (2020)

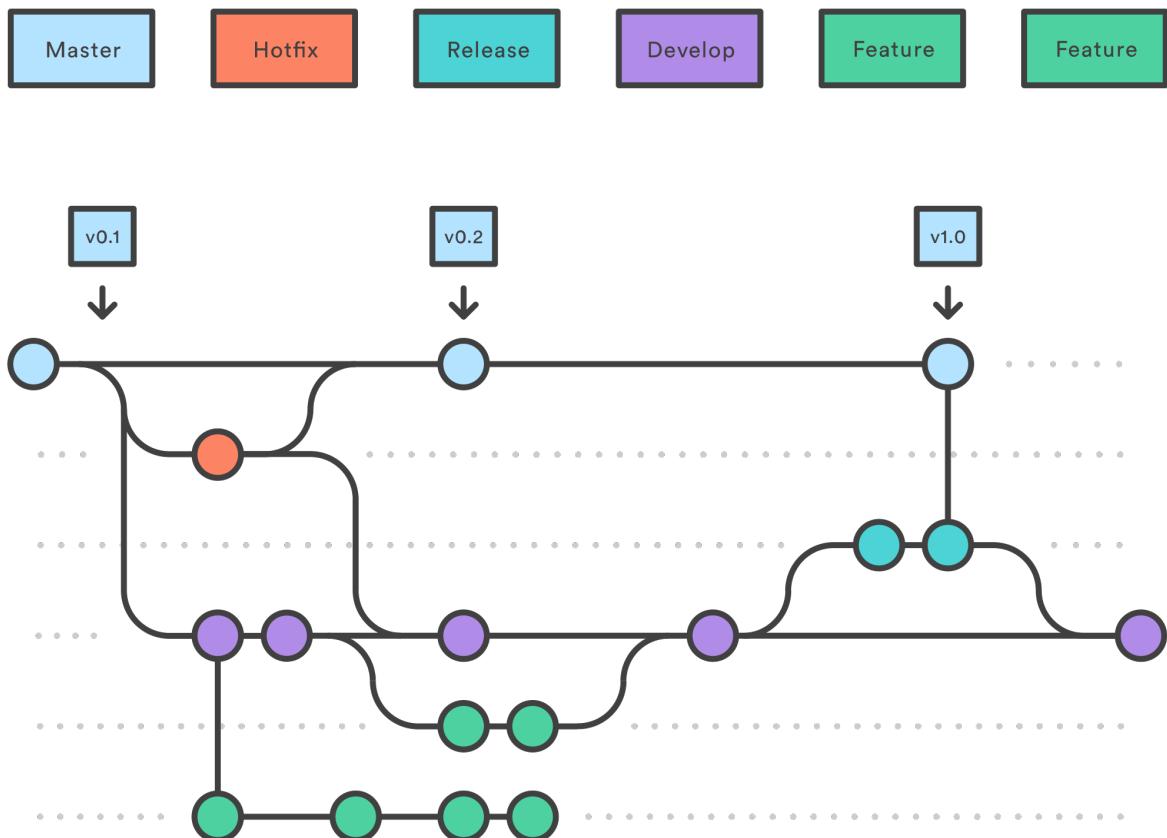


Abbildung 27: Gitflow³³

```
feature/backend/<konkretes-feature>
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 worden ist, meist ohne Probleme zusammengeführt werden.

6.8.1.1 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 Element wird an den ersten Platz geschoben, das erste Element an den zweiten Platz usw.) und `rollDown` (das unterste Element 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.

6.8.1.2 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 3: 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 wiederum 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 Argumenten 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.

6.8.1.3 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 Stapsels).

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 Element wird an den ersten Platz geschoben, das erste Element an den zweiten Platz usw.) und `rollDown` (das unterste Element 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.

6.8.1.4 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 4: Konzept für Nutzung generischer Schnittstelle

```
Operation.mit(matrixOperand, dezimalOperand, dezimal0Operand)
```

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 wiederum 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 Argumenten 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.

7 Dokumentation der sonstigen Beiträge der 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 Fazits aller Teammitglieder

8.1 Tom Bockhorn

8.2 Hendrik Falk

8.3 Dennis Gentges

8.4 Getuart Istogu

Rückblickend konnte ich aus dem Projekt viele neue Erkenntnisse gewinnen. Zumal war es mein erstes Projekt mit mehreren Beteiligten. Dadurch wurde mir bewusst, wie wichtig eine gute Aufgabenteilung und Absprache innerhalb eines Teams ist. Außerdem wurde mir durch das Projekt bewusst, wie wichtig eine gute Dokumentation ist, da ich für meine Entwicklung nachvollziehen können musste, was das Frontend gemacht hat, um bei der Implementierung der Menüführung auszuhelfen. Dazu habe ich bereits implementierten Code abstrahiert. Generell verlief das Projekt aus meiner Sicht gut. Wir konnten die Anforderungen fristgerecht erfüllen. Dabei konnte ich vor allem den Umgang von GitHub und Latex erlernen. Ich lernte Latex Wert zu schätzen, dadurch dass die Ausarbeitung ohne viel Mehraufwand sehr einheitlich aussah. Zuerst viel mir der Umgang damit jedoch schwer. Die Einarbeitung in GitHub war für mich auch sehr aufwändig, jedoch konnte ich von dem Wissen der anderen Teammitglieder profitieren.

8.5 Jannis Keienburg

8.6 Tim Jonas Meinerzhagen

8.7 Khang Pham

8.8 Tim Schwenke

9 Quellenverzeichnis

iju

10 Anhang - Quelltext

10.1 Model

10.1.1 Calculation

Listing 5: Action (Schwenke)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import de.fhdw.wip.rpntilecalculator.model.operands.Operand;

import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

import java.lang.reflect.InvocationTargetException;
import java.util.List;

/**
 * Summary: The framework for defining Actions. Actions are able to work
 *          ↪ with operands from the stack or executor functions.
 * Author: Tim Schwenke
 */
public abstract class Action {

    /**
     * Must be set by inheriting classes of {@link Action} for reflection
     *          ↪ to work.
     */
    protected Action scopedAction;

    /**
     * Must be overridden in case the required number of operands is a
     *          ↪ fixed amount.
     */
    protected int[] requiredNumOfOperands = new int[]{-1};

    /**
     * Leverages reflection for matching given arguments to a calculation
     *          ↪ method.
     *
     * @param operands List of operands.
     * @return Always a valid Operand.
     * @throws CalculationException In case the result cannot be computed.
     */
    @Contract(pure = true)
    public @NotNull Operand with(@NotNull List<Operand> operands) throws
        ↪ CalculationException {
        Operand[] target = new Operand[operands.size()];
        for (int i = 0; i < target.length; i++) {
            target[i] = operands.get(i);
        }
        return with(target);
    }
}

```

```


    /**
     * Leverages reflection for matching given arguments to a calculation
     * ↪ method.
     *
     * @param operands List of operands.
     * @return Always a valid Operand.
     * @throws CalculationException In case the result cannot be computed.
     */
    @Contract(pure = true) public @NotNull Operand with(@NotNull Operand...
        ↪ operands) throws CalculationException {
        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 (RuntimeException | NoSuchMethodException |
            ↪ IllegalAccessException | InvocationTargetException e) {
            throw new CalculationException(e.getMessage());
        }

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

    /**
     * @return Number of required operands for the concrete {@link Action}.
     * ↪ If {@code -1}
     * the number of operands required is variable.
     */
    public int[] getRequiredNumOfOperands() {
        return requiredNumOfOperands;
    }
}


```

Listing 6: ArcCosinus (Keienburg)

```


package de.fhdw.wip.rpntilecalculator.model.calculation;

import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import de.fhdw.wip.rpntilecalculator.model.operands.Operand;

/*


```

```

* Summary: Defines the arc Cosinus action.
* Author: Jannis Luca Keienburg
*/

public class ArcCosinus extends Action {

    @NotNull
    private static final ArcCosinus ARC_COSINUS = new ArcCosinus();

    /**
     * Singleton for COSINUS
     * @return singleton object
     */
    @Contract(pure = true) @NotNull public static ArcCosinus getInstance()
        ↪ { return ARC_COSINUS; }
    private ArcCosinus() {
        requiredNumOfOperands = new int[]{1};
    }

    @NotNull @Override
    public Operand with(@NotNull Operand... operands) throws
        ↪ CalculationException {
        scopedAction = this;
        return super.with(operands);
    }

    /**
     * Calculates the arc cosinus with a given angle.
     *
     * @param angle Representing the angle.
     * @return Result
     */
    @Contract(pure = true) @NotNull ODouble on(@NotNull ODouble angle) {
        ↪
        return new ODouble(Math.acos(Math.toRadians((angle.getDouble()))));
    }
}

```

Listing 7: ArcSinus (Keienburg)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import de.fhdw.wip.rpntilecalculator.model.operands.Operand;

/*
 * Summary: Defines the arc Sinus action.
 * Author: Jannis Luca Keienburg
 */

public class ArcSinus extends Action {

    @NotNull

```

```

private static final ArcSinus ARC_SINUS = new ArcSinus();

/*
 * Singleton for SINUS
 * @return singleton object
 */
@Contract(pure = true) @NotNull public static ArcSinus getInstance() {
    ↪ return ARC_SINUS; }

private ArcSinus() {
    requiredNumOfOperands = new int[]{1};
}

@NotNull @Override
public Operand with(@NotNull Operand... operands) throws
    ↪ CalculationException {
    scopedAction = this;
    return super.with(operands);
}

// Calculates the arc sinus with a given angle.
@Contract(pure = true) @NotNull ODouble on(@NotNull ODouble angle) {
    return new ODouble(Math.asin(Math.toRadians((angle.getDouble()))));
    ↪
}
}

```

Listing 8: ArcTangens (Keienburg)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import de.fhdw.wip.rpntilecalculator.model.operands.Operand;

/*
 * Summary: Defines the arc Tangens action.
 * Author: Jannis Luca Keienburg
 */

public class ArcTangens extends Action {

    @NotNull
    private static final ArcTangens ARC_TANGENS = new ArcTangens();

    /*
     * Singleton for SINUS
     * @return singleton object
     */
    @Contract(pure = true) @NotNull public static ArcTangens getInstance()
        ↪ { return ARC_TANGENS; }

    private ArcTangens() {
        requiredNumOfOperands = new int[]{1};
    }
}

```

```

    @NotNull @Override
    public Operand with(@NotNull Operand... operands) throws
        ↪ CalculationException {
        scopedAction = this;
        return super.with(operands);
    }

    // Calculates the tangens with a given angle.
    @Contract(pure = true) @NotNull ODouble on(@NotNull ODouble angle) {
        return new ODouble(Math.atan(Math.toRadians((angle.getDouble()))));
        ↪
    }
}

```

Listing 9: CalculationException (Schwenke)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

/**
 * Summary: Main Exception for a failed Calculation
 * Author: Tim Schwenke
 */
public class CalculationException extends Exception {

    /**
     * Create a new Exception
     * @param msg Error Message
     */
    public CalculationException(String msg) {
        super(msg);
    }

    /**
     * Create a new standard Exception
     */
    public CalculationException() {
        this("Not supported");
    }
}

```

Listing 10: Cosinus (Keienburg)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import de.fhdw.wip.rpntilecalculator.model.operands.Operand;

import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

/*
 * Summary: Defines the Cosinus action.
 * Author: Jannis Luca Keienburg
 */
@SuppressWarnings("unused")

```

```

public class Cosinus extends Action {

    @NotNull
    private static final Cosinus COSINUS = new Cosinus();

    /*
     * Singleton for COSINUS
     * @return singleton object
     */
    @Contract(pure = true) @NotNull public static Cosinus getInstance() {
        ↪ return COSINUS; }

    private Cosinus() {
        requiredNumOfOperands = new int[]{1, 2};
    }

    @NotNull @Override
    public Operand with(@NotNull Operand... operands) throws
        ↪ CalculationException {
        scopedAction = this;
        return super.with(operands);
    }

    // Calculates the cosinus with a given angle.
    @Contract(pure = true) @NotNull ODouble on(@NotNull ODouble angle) {
        return new ODouble(Math.cos(Math.toRadians((angle.getDouble()))));
    }
}

```

Listing 11: Derivation (Keienburg)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import org.apache.commons.math3.analysis.polynomials.PolynomialFunction;

import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

import de.fhdw.wip.rpntilecalculator.model.operands.OPolynom;
import de.fhdw.wip.rpntilecalculator.model.operands.Operand;

/*
 * Summary: A Class that can calculate the derivate of a function
 * Author: Jannis Luca Keienburg
 */

public class Derivation extends Action {

    @NotNull private static final Derivation DERIVATION = new Derivation()
        ↪ ;

    @Contract(pure = true) @NotNull public static Derivation getInstance()
        ↪ { return DERIVATION; }

    private Derivation() {
        requiredNumOfOperands = new int[]{1};
    }
}

```

```

    @NotNull @Override
    public Operand with(@NotNull Operand... operands) throws
        ↪ CalculationException {
        scopedAction = this;
        return super.with(operands);
    }

    @Contract(pure = true) @NotNull OPolynom on(@NotNull OPolynom oPolynom
        ↪ ) {
        return derivate(oPolynom);
    }

    // Structure: via the method getCoefficients()
    // coefficients[n] * x^n + ... + coefficients[1] * x + coefficients[0]

    // Sets the Function into the following formatition:
    // [n]* x^10 + [n-1] *x^9 + ... + [1] * x + [0]
    private double[] getFunctionAsDouble(@NotNull OPolynom oPolynom) {
        return oPolynom.getPolynom().getCoefficients();
    }

    // Calculates the derivation
    // returns it as an OPolynom
    public OPolynom derivate(OPolynom oPolynom)
    {
        double[] function = getFunctionAsDouble(oPolynom);
        double[] derivation = new double[function.length - 1];

        for(int i = function.length - 1; i > 0; i--)
        {
            derivation[i-1] = function[i] * i;
        }
        return new OPolynom(new PolynomialFunction(derivation));
    }
}

```

Listing 12: HighAndLowPoints (Keienburg)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import org.apache.commons.math3.analysis.polynomials.PolynomialFunction;

import org.intellij.lang.annotations.JdkConstants;
import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

import de.fhdw.wip.rpntilecalculator.model.operands.OPolynom;
import de.fhdw.wip.rpntilecalculator.model.operands.OTuple;
import de.fhdw.wip.rpntilecalculator.model.operands.Operand;

/*
 * Summary: A Class that can calculate the high and the low points of a
 *          ↪ function( up to third grade)
 * Author: Jannis Luca Keienburg
 */

```

```

public class HighAndLowPoints extends Action {

    @NotNull private static final HighAndLowPoints HIGH_AND_LOW_POINTS =
        ↪ new HighAndLowPoints();
    @NotNull private static final Derivation DERIVATION = Derivation.
        ↪ getInstance();
    @NotNull private static final Zeros ZEROS = Zeros.getInstance();

    @Contract(pure = true) @NotNull public static HighAndLowPoints
        ↪ getInstance() { return HIGH_AND_LOW_POINTS; }

    private HighAndLowPoints() {
        requiredNumOfOperands = new int[]{1};
    }

    @NotNull @Override
    public Operand with(@NotNull Operand... operands) throws
        ↪ CalculationException {
        scopedAction = this;
        return super.with(operands);
    }

    @Contract(pure = true) @NotNull OTuple on(@NotNull OPolynom oPolynom)
        ↪ {
        return new OTuple(getHighAndLowPoints(oPolynom));
    }

    // Begin. Returns an double array with the following structure
    public double [] getHighAndLowPoints(OPolynom oPolynom)
    {
        double [] result = calculateHighAndLowPoints(oPolynom);
        return result;
    }

    // Sets the Function into the following formatition:
    // [n]* x^10 + [n-1] *x^9 + ... + [1] * x + [0]
    private double[] getFunctionAsDouble(OPolynom oPolynom) {
        return oPolynom.getPolynom().getCoefficients();
    }

    // Calculates the values of the extreme points of a given function.
    // Returns the values as an double array, an uneyal number position
        ↪ stands for the x value,
    // the number at the next position for the y value
    private double[] calculateHighAndLowPoints(OPolynom oPolynom)
    {
        // Calculates the derivation of the funcion.
        OPolynom derivation = DERIVATION.derivate(oPolynom);
        // Calculates the zeros of the function's derivation.
        double [] zeros = ZEROS.calculateZeros(derivation);
        // Gets the funcion as an double array for the calculation.
        double [] functionAsDouble = getFunctionAsDouble(oPolynom);

        // Calculates the y values of the zeros.
    }
}

```

```

        double [] valuesXY = new double[] {};
        int position = -1;
        for(int counter = 0; counter < zeros.length; counter++)
        {
            double currentZero = zeros[counter];
            double yValue = 0;
            for(int counter2 = 0; counter < functionAsDouble.length;
                ↪ counter2++)
            {
                yValue += functionAsDouble[counter] * Math.pow(currentZero
                    ↪ ,(functionAsDouble.length - counter2 - 1));
            }
            position++;
            valuesXY[position] = currentZero;
            position++;
            valuesXY[position] = yValue;
        }
        return valuesXY;
    }
}

```

Listing 13: Integral (Istogu)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import org.apache.commons.math3.analysis.UnivariateFunction;
import org.apache.commons.math3.analysis.integration.SimpsonIntegrator;
import org.apache.commons.math3.analysis.polynomials.PolynomialFunction;
import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import de.fhdw.wip.rpntilecalculator.model.operands.OPolynom;
import de.fhdw.wip.rpntilecalculator.model.operands.Operand;

/*
 * Summary: Calculate the antiderivative and integral
 * Author: Getuart Istogu
 */

public class Integral extends Action {

    @NotNull private static final Integral INTEGRAL = new Integral();

    @Contract(pure = true) @NotNull public static Integral getInstance() {
        ↪ return INTEGRAL; }

    private Integral() {requiredNumOfOperands = new int[] {3};}

    @NotNull @Override
    public Operand with(@NotNull Operand... operands) throws
        ↪ CalculationException {
        scopedAction = this;
        return super.with(operands);
    }

    @Contract(pure = true) @NotNull ODouble on(@NotNull OPolynom oPolynom,

```

```

    ↪ @NotNull ODouble lowerBound, @NotNull ODouble upperBound) {
    return calculateIntegralSimpsons(oPolynom, lowerBound.getDouble(),
        ↪ upperBound.getDouble());
}

/**
 * Calculates the integral for the specified limit range
 * @param oPolynom Normal PolynomialFunction, not its antiderivative
 * @param lowerBound Lower limit
 * @param upperBound Upper limit
 * @return
 */
@NotNull
public ODouble calculateIntegralSimpsons(OPolynom oPolynom, double
    ↪ lowerBound, double upperBound)
{
    SimpsonIntegrator simpsonIntegrator = new SimpsonIntegrator();
    UnivariateFunction uF = (UnivariateFunction) oPolynom.getPolynom();
    ↪
    return new ODouble(simpsonIntegrator.integrate(10000, uF,
        ↪ lowerBound, upperBound));
}

/**
 * Calculates the antiderivative of the passed function
 * @param oPolynom Examined function
 * @return Its antiderivative
 */
public OPolynom getAntiderivative(OPolynom oPolynom)
{
    PolynomialFunction polynomialFunction = oPolynom.getPolynom();

    //  $3x^2 - 2x + 5$  == 3rd degree, but 4 coefficients
    // Therefore number of coefficient antiderivative = degree + 2
    int degreeForAntiderivative = polynomialFunction.degree() + 2;

    double[] functionCoefficients = polynomialFunction.getCoefficients()
        ↪ ;

    double[] antiderivativeCoefficients = new double[
        ↪ degreeForAntiderivative];

    // This value can be of any size. It is referred as "C" in the
    ↪ literature.
    antiderivativeCoefficients[0] = 0;
    for(int i = 1; i < degreeForAntiderivative; i++)
    {
        antiderivativeCoefficients[i] = functionCoefficients[i-1]/(
            ↪ double) i;
    }

    return new OPolynom(new PolynomialFunction(
        ↪ antiderivativeCoefficients));
}
}

```

Listing 14: IntegralTest (Istogu)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import org.apache.commons.math3.analysis.polynomials.PolynomialFunction;
import org.junit.Test;

import de.fhdw.wip.rpntilecalculator.model.operands.OPolynom;

import static org.junit.Assert.*;

/*
 * Summary: Test for the class 'Integral'
 * Author: Getuart Istogu
 */

public class IntegralTest {
    private Integral INTEGRAL = Integral.getInstance();

    @Test public void getSimpsonIntegrator_isCorrect() {
        double[] inputCoefficients = new double[]{17, -8, 1};
        OPolynom inputFunction = new OPolynom(new PolynomialFunction(
            ↪ inputCoefficients));

        assertEquals(INTEGRAL.calculateIntegralSimpsons
            (inputFunction, 2, 5).getDouble(), 6, 0.001);
    }

    @Test
    public void getAntiderivative_isCorrect() {

        OPolynom antiderivative = INTEGRAL.getAntiderivative(new OPolynom(
            ↪ new double[] {
                7, 3, 0, -3.21
            }));
        double[] expectedCoefficient = new double[] {
            0, 7, 1.5, 0, -0.8025
        };

        double[] coefficient = antiderivative.getPolynom().getCoefficients
            ↪ ();

        assertEquals(coefficient[0], expectedCoefficient[0], 0.001);
        assertEquals(coefficient[1], expectedCoefficient[1], 0.001);
        assertEquals(coefficient[2], expectedCoefficient[2], 0.001);
        assertEquals(coefficient[3], expectedCoefficient[3], 0.001);
        assertEquals(coefficient[4], expectedCoefficient[4], 0.001);
    }
}

```

Listing 15: Limes (Istogu)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import org.apache.commons.math3.analysis.polynomials.PolynomialFunction;

```

```

import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import de.fhdw.wip.rpntilecalculator.model.operands.OPolynom;
import de.fhdw.wip.rpntilecalculator.model.operands.Operand;

/*
 * Summary: Determine limit
 * Author: Getuart Istogu
 */
public class Limes extends Action {

    @NotNull private static final Limes LIMES = new Limes();

    @Contract(pure = true) @NotNull public static Limes getInstance() {
        ↪ return LIMES; }

    private Limes() { requiredNumOfOperands = new int[] {2}; }

    @NotNull @Override
    public Operand with(@NotNull Operand... operands) throws
        ↪ CalculationException {
        scopedAction = this;
        return super.with(operands);
    }

    @Contract(pure = true) @NotNull ODouble on (@NotNull OPolynom oPolynom
        ↪ , @NotNull ODouble approach) {
        return limit(oPolynom, approach.getDouble());
    }

    /**
     * Calculates the limit of a function at one point
     * @param oPolynom Examined function
     * @param approach Limit point
     * @return Limit value at the defined point
     */
    public ODouble limit(OPolynom oPolynom, double approach) {
        PolynomialFunction polynomialFunction = oPolynom.getPolynom();
        double below = limitFromBelow(polynomialFunction, approach);
        double above = limitFromAbove(polynomialFunction, approach);
        if(below == above)
            return new ODouble(below);
        else
            return new ODouble(Double.NaN);
    }

    /**
     * Calculates the limit of a function at one point from below
     * @param polynomialFunction Examined function
     * @param approach Limit point
     * @return Limit value from below at the defined point
     */
    private double limitFromBelow(PolynomialFunction polynomialFunction,

```

```

    ↪ double approach) {

        for (double d = approach - 10; d <= approach; d = approach
            - ((approach - d) / 10)) {
            if (polynomialFunction.value(d) == Double.POSITIVE_INFINITY) {
                return Double.POSITIVE_INFINITY;
            } else if (polynomialFunction.value(d) == Double.
                ↪ NEGATIVE_INFINITY) {
                return Double.NEGATIVE_INFINITY;
            } else if (Double.isNaN(polynomialFunction.value(d))) {
                return polynomialFunction.value(approach + ((approach - d)
                    ↪ * 10));
            } else {
                if (d == approach) {
                    return polynomialFunction.value(d);
                } else if (approach - d < 0.0000000001) {
                    d = approach;
                }
            }
        }
        return Double.NaN;
    }

    /**
     * Calculates the limit of a function at one point from above
     * @param polynomialFunction Examined function
     * @param approach Limit point
     * @return Limit value from above at the defined point
     */
    private double limitFromAbove(PolynomialFunction polynomialFunction,
        ↪ double approach) {

        for (double d = approach + 10; d >= approach; d = approach
            - ((approach - d) / 10)) {
            if (polynomialFunction.value(d) == Double.POSITIVE_INFINITY) {
                return Double.POSITIVE_INFINITY;
            } else if (polynomialFunction.value(d) == Double.
                ↪ NEGATIVE_INFINITY) {
                return Double.NEGATIVE_INFINITY;
            } else if (Double.isNaN(polynomialFunction.value(d))) {
                return polynomialFunction.value(approach + ((approach - d)
                    ↪ * 10));
            } else {
                if (d == approach) {
                    return polynomialFunction.value(d);
                } else if (d - approach < 0.0000000001) {
                    d = approach;
                }
            }
        }
        return Double.NaN;
    }
}

```

Listing 16: LimesTest (Istogu)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import org.junit.Test;

import de.fhdw.wip.rpntilecalculator.model.operands.OPolynom;

import static org.junit.Assert.assertEquals;

/*
 * Summary: Test for the class 'Limes'
 * Author: Getuart Istogu
 */

public class LimesTest {
    private Limes LIMES = Limes.getInstance();

    @Test
    public void limes_isCorrect1()
    {
        OPolynom polynom = new OPolynom(new double[]{1, 0, 1});
        assertEquals(
            LIMES.limit(polynom, 5).getDouble(), 26, 0.001);
    }

    @Test
    public void limes_isCorrect2()
    {
        OPolynom polynom = new OPolynom(new double[]{0, 0, 2, -0.333333333
            ↪ });
        assertEquals(
            LIMES.limit(polynom, Double.POSITIVE_INFINITY).getDouble(),
            ↪ Double.NEGATIVE_INFINITY, 0.001);
    }
}

```

Listing 17: Logarithm (Keienburg)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import de.fhdw.wip.rpntilecalculator.model.operands.Operand;

/*
 * Summary: A Class that can calculate the natural logarithm.
 * Author: Jannis Luca Keienburg
 */
public class Logarithm extends Action {

    @NotNull
    private static final Logarithm LOGARITHM = new Logarithm();

```

```

/*
 * Singleton for Logarithm
 * @return singleton object
 */
@Contract(pure = true) @NotNull public static Logarithm getInstance()
    ↪ { return LOGARITHM; }
private Logarithm() {
    requiredNumOfOperands = new int[]{1, 2};
}

@NotNull @Override
public Operand with(@NotNull Operand... operands) throws
    ↪ CalculationException {
    scopedAction = this;
    return super.with(operands);
}

//Natural Logarithm
@Contract(pure = true) @NotNull ODouble on(@NotNull ODouble oDouble) {
    if(oDouble.getDouble() <= 0)
        throw new IllegalArgumentException("Value must be higher than
            ↪ Zero.");
    return new ODouble(Math.log(oDouble.getDouble()));
}

//Logarithm with specific base
@Contract(pure = true) @NotNull ODouble on(@NotNull ODouble base,
    ↪ ODouble logartihmOf) {
    if(base.getDouble() <= 0 || logartihmOf.getDouble() <= 0)
        throw new IllegalArgumentException("Value must be higher than
            ↪ Zero.");
    return new ODouble(Math.log(logartihmOf.getDouble()) / Math.log(
        ↪ base.getDouble()));
}
}

```

Listing 18: Logarithm10 (Keienburg)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;
import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import de.fhdw.wip.rpntilecalculator.model.operands.Operand;

/*
 * Summary: A Class that can calculate the natural logarithm.
 * Author: Jannis Luca Keienburg
 */
public class Logarithm10 extends Action {

    @NotNull
    private static final Logarithm10 LOGARITHM10 = new Logarithm10();
}

```

```

/*
 * Singleton for Logarithm with the base of 10
 * @return singleton object
 */
@Contract(pure = true) @NotNull public static Logarithm10 getInstance
    () { return LOGARITHM10; }
private Logarithm10() {
    requiredNumOfOperands = new int[]{1};
}

@NotNull @Override
public Operand with(@NotNull Operand... operands) throws
    CalculationException {
    scopedAction = this;
    return super.with(operands);
}

//Logarithm to the base of 10
@Contract(pure = true) @NotNull ODouble on(@NotNull ODouble oDouble) {
    if(oDouble.getDouble() <= 0)
        throw new IllegalArgumentException("Value must be higher than
            Zero.");
    return new ODouble(Math.log10(oDouble.getDouble()));
}
}

```

Listing 19: MatrixUtil (Istogu)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

import de.fhdw.wip.rpntilecalculator.model.operands.OMatrix;
import de.fhdw.wip.rpntilecalculator.model.operands.OMSet;
import de.fhdw.wip.rpntilecalculator.model.operands.OTuple;
import de.fhdw.wip.rpntilecalculator.model.operands.Operand;

/*
 * Summary: Solving systems of linear equations with "LR decomposition
 *           with column pivot search"
 * Author: Getuart Istogu
 */

public class MatrixUtil extends Action {
    @NotNull
    private static final MatrixUtil MATRIX_UTIL = new MatrixUtil();

    @Contract(pure = true) @NotNull public static MatrixUtil getInstance()
        { return MATRIX_UTIL; }
    private MatrixUtil() { requiredNumOfOperands = new int[] {2}; }

    @NotNull @Override
    public Operand with(@NotNull Operand... operands) throws
        CalculationException {
        scopedAction = this;
    }
}

```

```

        return super.with(operands);
    }

@Contract(pure = true) @NotNull OTuple on (@NotNull OMMatrix A, OTuple
    ↪ b) {
    return solveLinearSystem(A, b.getTuple());
}
/***
 * On the condition that  $A \cdot x = b$ 
 * @param A Matrix
 * @param b Solution vector
 * @return Returns the vector 'x'
 */
public OTuple solveLinearSystem(OMatrix A, double[] b)
{
    return new OTuple(solveLGSForX(A.getMatrix().getData(), b));
}

/***
 * Determines pivot vector
 * @param A The linear system in double[][][]
 * @return
 */
private int[] pivot(double A[][])
{
    int n = A.length;
    int[] pivot = new int[n];
    for (int j = 0; j < n-1; j++)
    {
        double max = Math.abs(A[j][j]);
        int imax = j;
        for (int i = j+1; i < n; i++)
            if (Math.abs(A[i][j]) > max)
            {
                max = Math.abs(A[i][j]);
                imax = i;
            }
        double[] h = A[j];
        A[j] = A[imax];
        A[imax] = h;
        pivot[j] = imax;
        for (int i = j+1; i < n; i++)
        {
            double f = -A[i][j]/A[j][j];
            for (int k = j+1; k < n; k++)
                A[i][k] += f*A[j][k];
            A[i][j] = -f;
        }
    }
    return pivot;
}

/***
 * Loest das LGS  $A \cdot x = b$  nach x auf

```

```

 * @param A The linear system in double[][][]
 * @param b Solution vector
 * @return Returns the vector 'x'
 */
private double[] solveLGSForX(double[][] A, double[] b)
{
    double[][] B = A.clone();
    double[] x = b.clone();
    int[] pivot = pivot(B);
    int n = B.length;
    for (int i = 0; i < n-1; i++)
    {
        double h = b[pivot[i]];
        b[pivot[i]] = b[i];
        b[i] = h;
    }
    for (int j = 0; j < n; j++)
    {
        x[j] = b[j];
        for (int i = 0; i < j; i++)
            x[j] -= B[j][i]*x[i];
    }
    for (int j = n-1; j >= 0; j--)
    {
        for (int k = j+1; k < n; k++)
            x[j] -= B[j][k]*x[k];
        x[j] /= B[j][j];
    }
    return x;
}
}

```

Listing 20: MatrixUtilTest (Istogu)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import org.junit.Test;

import de.fhdw.wip.rpntilecalculator.model.operands.OMatrix;

import static org.junit.Assert.assertEquals;

/*
 * Summary: Test for the class 'MatrixUtil'
 * Author: Getuart Istogu
 */

public class MatrixUtilTest {
    private MatrixUtil MATRIX_UTIL = MatrixUtil.getInstance();

    @Test
    public void solveLinearSystem_isCorrect() {
        double[][] A = {
            {3, -7, 0, -6},
            {2, -8, -1, 4},

```

```

    {0, 9, -7, 9},
    {-4, 5, -3, -8}};

OMatrix aMatrix = new OMatrix(A);
double[] b = {8, 1, -7, -1};

double[] exptectedX = {1.1970139565076277, -0.12852969814995122,
    ↪ 0.08276533592989294, -0.5848750405712432};

double[] x = MATRIX_UTIL.solveLinearSystem(aMatrix, b).getTuple();

assertEquals(x[0], exptectedX[0], 0.001);
assertEquals(x[1], exptectedX[1], 0.001);
assertEquals(x[2], exptectedX[2], 0.001);
assertEquals(x[3], exptectedX[3], 0.001);
}

}

```

Listing 21: Minus (Falk)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import de.fhdw.wip.rpntilecalculator.model.operands.OFraction;
import de.fhdw.wip.rpntilecalculator.model.operands.OMatrix;
import de.fhdw.wip.rpntilecalculator.model.operands.OPolynom;
import de.fhdw.wip.rpntilecalculator.model.operands.OSet;
import de.fhdw.wip.rpntilecalculator.model.operands.OTuple;
import de.fhdw.wip.rpntilecalculator.model.operands.Operand;

import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

/*
 * Summary: Defines the Minus click. Lets the user subtract operands.
 * Author: Hendrik Falk
 */
public class Minus extends Action {

    @NotNull private static final Plus PLUS = Plus.getInstance();

    @NotNull private static final Minus MINUS = new Minus();

    @Contract(pure = true) @NotNull public static Minus getInstance() {
        ↪ return MINUS; }

    private Minus() {
        requiredNumOfOperands = new int[]{2};
    }

    @NotNull @Override
    public Operand with(@NotNull Operand... operands) throws
        ↪ CalculationException {
        scopedAction = this;
        return super.with(operands);
    }
}

```

```

//region Double
//
//      ↪ -----
//      ↪

@Contract(pure = true) @NotNull ODouble on(@NotNull ODouble oDouble1,
    ↪ @NotNull ODouble oDouble2) {
    return new ODouble(oDouble1.getDouble() - oDouble2.getDouble());
}

@Contract(pure = true) @NotNull ODouble on(@NotNull ODouble oDouble,
    ↪ @NotNull OFraction oFraction) {
    return new ODouble(oDouble.getDouble() - oFraction.getDouble());
}

//endregion

//region Fraction
//
//      ↪ -----
//      ↪

@Contract(pure = true) @NotNull OFraction on(@NotNull OFraction
    ↪ oFraction1, @NotNull OFraction oFraction2) {
    return new OFraction(oFraction1.getFraction().subtract(oFraction2.
        ↪ getFraction()));
}

@Contract(pure = true) @NotNull ODouble on(@NotNull OFraction
    ↪ oFraction, @NotNull ODouble oDouble) {
    return new ODouble(oFraction.getDouble() - oDouble.getDouble());
}

//endregion

//region Set
//
//      ↪ -----
//      ↪

@Contract(pure = true) @NotNull OSet on(@NotNull OSet oSet, @NotNull
    ↪ ODouble oDouble) {
    return PLUS.on(oDouble.turnAroundSign(), oSet);
}

@Contract(pure = true) @NotNull OSet on(@NotNull OSet oSet, @NotNull
    ↪ OFraction oFraction) {
    return on(oSet, new ODouble(oFraction.getDouble()));
}

//endregion

//region Matrix
//

```

```

    ↵ -----
    ↵

@Contract(pure = true) @NotNull OMatrix on(@NotNull OMatrix oMatrix1,
    ↵ @NotNull OMatrix oMatrix2) {
    return new OMatrix(oMatrix1.getMatrix().subtract(oMatrix2.
        ↵ getMatrix()));
}

@Contract(pure = true) @NotNull OMatrix on(@NotNull OMatrix oMatrix,
    ↵ @NotNull ODouble oDouble) {
    return new OMatrix(oMatrix.getMatrix().scalarAdd(oDouble.
        ↵ turnAroundSign().getDouble()));
}

@Contract(pure = true) @NotNull OMatrix on(@NotNull OMatrix oMatrix,
    ↵ @NotNull OFraction oFraction) {
    return new OMatrix(oMatrix.getMatrix().scalarAdd(oFraction.
        ↵ turnAroundSign().getDouble()));
}

//endregion

//region Polynom
//
    ↵ -----
    ↵

@Contract(pure = true) @NotNull OPolynom on(@NotNull OPolynom
    ↵ oPolynom1, @NotNull OPolynom oPolynom2) {
    return new OPolynom(oPolynom1.getPolynom().subtract(oPolynom2.
        ↵ getPolynom()));
}

@Contract(pure = true) @NotNull OPolynom on(@NotNull OPolynom oPolynom
    ↵ , @NotNull ODouble oDouble) {
    return PLUS.on(oDouble.turnAroundSign(), oPolynom);
}

@Contract(pure = true) @NotNull OPolynom on(@NotNull OPolynom oPolynom
    ↵ , @NotNull OFraction oFraction) {
    return on(oPolynom, new ODouble(oFraction.getDouble()));
}

//endregion

//region Tuple
//
    ↵ -----
    ↵

@Contract(pure = true) @NotNull OTuple on(@NotNull OTuple oTuple1,
    ↵ @NotNull OTuple oTuple2) {
    return PLUS.on(oTuple1, oTuple2.turnAroundSign());
}

```

```

@Contract(pure = true) @NotNull OTuple on(@NotNull OTuple oTuple,
    ↪ @NotNull ODouble oDouble) {
    return PLUS.on(oDouble.turnAroundSign(), oTuple);
}

@Contract(pure = true) @NotNull OTuple on(@NotNull OTuple oTuple,
    ↪ @NotNull OFraction oFraction) {
    return PLUS.on(oFraction.turnAroundSign(), oTuple);
}

//endregion
}

```

Listing 22: MinusTest (Schwenke)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import org.junit.Test;

import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import de.fhdw.wip.rpntilecalculator.model.operands.OFraction;
import de.fhdw.wip.rpntilecalculator.model.operands.OMatrix;
import de.fhdw.wip.rpntilecalculator.model.operands.OPolynom;
import de.fhdw.wip.rpntilecalculator.model.operands.OSet;
import de.fhdw.wip.rpntilecalculator.model.operands.OTuple;

import static org.junit.Assert.*;

/*
 * Summary: Test for the class 'MinusTest'
 * Author: Tim Schwenke
 */

public class MinusTest {

    private Minus MINUS = Minus.getInstance();

    @Test public void on_DoubleDouble_isCorrect() {
        System.out.println(MINUS.on(
            new ODouble(5),
            new ODouble(5)
        ));

        assertTrue(MINUS.on(
            new ODouble(5),
            new ODouble(5)
        ).equalsValue(new ODouble(0)));
    }

    @Test public void on_DoubleFraction_isCorrect() {
        assertTrue(MINUS.on(
            new ODouble(5),
            new OFraction(1, 2)
        ).equalsValue(new ODouble(5)));
    }
}

```

```

        ).equalsValue(new ODouble(4.5)));
    }

@Test public void on_FractionFraction_isCorrect() {
    assertTrue(MINUS.on(
        new OFraction(1, 1),
        new OFraction(1, 1)
    ).equalsValue(new OFraction(0)));
}

@Test public void on_FractionDouble_isCorrect() {
    assertTrue(MINUS.on(
        new OFraction(1, 1),
        new ODouble(1)
    ).equalsValue(new ODouble(0)));
}

@Test public void on_SetDouble_isCorrect() {
    assertTrue(MINUS.on(
        new OSet(1, 2),
        new ODouble(1)
    ).equalsValue(new OSet(0, 1)));
}

@Test public void on_SetFraction_isCorrect() {
    assertTrue(MINUS.on(
        new OSet(1, 2),
        new OFraction(1, 1)
    ).equalsValue(new OSet(0, 1)));
}

@Test public void on_MatrixMatrix_isCorrect() {
    assertTrue(MINUS.on(
        new OMatrix(new double[][]{
            {1d, 2d, 3d},
            {1d, 2d, 3d},
            {1d, 2d, 3d}
        }),
        new OMatrix(new double[][]{
            {1d, 2d, 3d},
            {1d, 2d, 3d},
            {1d, 2d, 3d}
        })
    ).equalsValue(new OMatrix(new double[][]{
        {0d, 0d, 0d},
        {0d, 0d, 0d},
        {0d, 0d, 0d}
    })));
}

@Test public void on_MatrixMatrix_isWrongDimension() {
    try {
        MINUS.on(
            new OMatrix(new double[][]{

```

```

        {1d, 2d, 3d},
        {1d, 2d, 3d},
        {1d, 2d, 3d}
    }),
    new OMatrix(new double[][]{
        {1d, 1d},
        {2d, 3d}
    })
);
fail();
} catch (RuntimeException e) {
    assertTrue("Should always fail", true);
}
}

@Test public void on_MatrixDouble_isCorrect() {
    assertTrue(MINUS.on(
        new OMatrix(new double[][]{
            {1, 2, 3},
            {1, 2, 3}
        }),
        new ODouble(1)
    ).equalsValue(new OMatrix(new double[][]{
        {0, 1, 2},
        {0, 1, 2}
    })));
}

@Test public void on_MatrixFraction_isCorrect() {
    assertTrue(MINUS.on(
        new OMatrix(new double[][]{
            {1, 2, 3},
            {1, 2, 3}
        }),
        new OFraction(1, 1)
    ).equalsValue(new OMatrix(new double[][]{
        {0, 1, 2},
        {0, 1, 2}
    })));
}

@Test public void onPolynomPolynom_isCorrect() {
    assertTrue(MINUS.on(
        new OPolynom(1, 2, 3),
        new OPolynom(2, 0, 5, 6)
    ).equalsValue(new OPolynom(-1, 2, -2, -6)));
}

@Test public void on_PolynomFraction_isCorrect() {
    assertTrue(MINUS.on(
        new OPolynom(1, 2, 3),
        new OFraction(1, 1)
    ).equalsValue(new OPolynom(0, 2, 3)));
}
}

```

```

@Test public void on_PolynomDouble_isCorrect() {
    assertTrue(MINUS.on(
        new OPolynom(1, 2, 3),
        new ODouble(1)
    ).equalsValue(new OPolynom(0, 2, 3)));
}

@Test public void on_TupleTuple_isCorrect() {
    assertTrue(MINUS.on(
        new OTuple(1, 2),
        new OTuple(1, 2)
    ).equalsValue(new OTuple(0, 0)));
}

@Test public void on_TupleDouble_isCorrect() {
    assertTrue(MINUS.on(
        new OTuple(1, 2),
        new ODouble(1)
    ).equalsValue(new OTuple(0, 1)));
}

@Test public void on_TupleFraction_isCorrect() {
    assertTrue(MINUS.on(
        new OTuple(1, 2),
        new OFraction(1, 1)
    ).equalsValue(new OTuple(0, 1)));
}
}

```

Listing 23: Modulo (Falk)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import de.fhdw.wip.rpntilecalculator.model.operands.Operand;

import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Contract;

/*
 * Summary: Defines the Modulo click.
 * Author: Hendrik Falk
 */
public class Modulo extends Action {
    @NotNull private static final Modulo MODULO = new Modulo();

    @Contract(pure = true) @NotNull public static Modulo getInstance() {
        ↪ return MODULO; }

    private Modulo() {
        requiredNumOfOperands = new int[]{2};
    }

    @NotNull @Override
    public Operand with(@NotNull Operand... operands) throws

```

```

    ↗ CalculationException {
        scopedAction = this;
        return super.with(operands);
    }

    //region Integer
    //
    ↗ -----
    ↗
    /*
     * Modulo operations. It should be noted that normally it isn't allow
     * ↗ to modulo with floating numbers.
     * However it is possible in Java.
     * @return result of the modulo operations
     */
    @Contract(pure = true) @NotNull ODouble on(@NotNull ODouble dividend,
        ↗ @NotNull ODouble divisor) {
        return new ODouble(dividend.getDouble() % divisor.getDouble());
    }
}

```

Listing 24: Plus (Schwenke)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import de.fhdw.wip.rpntilecalculator.model.operands.OFraction;
import de.fhdw.wip.rpntilecalculator.model.operands.OMatrix;
import de.fhdw.wip.rpntilecalculator.model.operands.OPolynom;
import de.fhdw.wip.rpntilecalculator.model.operands.OSet;
import de.fhdw.wip.rpntilecalculator.model.operands OTuple;
import de.fhdw.wip.rpntilecalculator.model.operands.Operand;

import org.apache.commons.math3.analysis.polynomials.PolynomialFunction;
import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

import java.util.HashSet;
import java.util.Set;

/*
 * Summary: Defines the Plus click. Lets the user subtract operands.
 * Author: Tim Schwenke
 */
public class Plus extends Action {

    @NotNull private static final Plus PLUS = new Plus();

    @Contract(pure = true) @NotNull public static Plus getInstance() {
        ↗ return PLUS; }
    private Plus() {
        requiredNumOfOperands = new int[]{2};
    }

    @NotNull @Override
    public Operand with(@NotNull Operand... operands) throws

```

```

    ↪ CalculationException {
    scopedAction = this;
    return super.with(operands);
}

@Contract(pure = true) @NotNull ODouble on(@NotNull ODouble oDouble1,
    ↪ @NotNull ODouble oDouble2) {
    return new ODouble(oDouble1.getDouble() + oDouble2.getDouble());
}

@Contract(pure = true) @NotNull ODouble on(@NotNull ODouble oDouble,
    ↪ @NotNull OFraction oFraction) {
    return new ODouble(oDouble.getDouble() + oFraction.getDouble());
}

@Contract(pure = true) @NotNull ODouble on(@NotNull OFraction
    ↪ oFraction, @NotNull ODouble oDouble) {
    return on(oDouble, oFraction);
}

@Contract(pure = true) @NotNull OSet on(@NotNull ODouble oDouble,
    ↪ @NotNull OSet oSet) {
    Set<Double> newSet = new HashSet<>();
    for (double d : oSet.getSet())
        newSet.add(d + oDouble.getDouble());
    return new OSet(newSet);
}

@Contract(pure = true) @NotNull OSet on(@NotNull OSet oSet, @NotNull
    ↪ ODouble oDouble) {
    return on(oDouble, oSet);
}

@Contract(pure = true) @NotNull OMatrix on(@NotNull ODouble oDouble,
    ↪ @NotNull OMatrix oMatrix) {
    return new OMatrix(oMatrix.getMatrix().scalarAdd(oDouble.getDouble(
        ↪())));
}

@Contract(pure = true) @NotNull OMatrix on(@NotNull OMatrix oMatrix,
    ↪ @NotNull ODouble oDouble) {
    return on(oDouble, oMatrix);
}

@Contract(pure = true) @NotNull OPolynom on(@NotNull ODouble oDouble,
    ↪ @NotNull OPolynom oPolynom) {
    double[] d = oPolynom.getPolynom().getCoefficients();
    d[0] += oDouble.getDouble();
    return new OPolynom(new PolynomialFunction(d));
}

@Contract(pure = true) @NotNull OPolynom on(@NotNull OPolynom oPolynom
    ↪ , @NotNull ODouble oDouble) {
    return on(oDouble, oPolynom);
}

```

```

}

@Contract(pure = true) @NotNull OTuple on(@NotNull ODouble oDouble,
    ↪ @NotNull OTuple oTuple) {
    double[] oldTuple = oTuple.getTuple();
    double[] newTuple = new double[oldTuple.length];
    for (int i = 0; i < newTuple.length; i++)
        newTuple[i] = oDouble.getDouble() + oldTuple[i];
    return new OTuple(newTuple);
}

@Contract(pure = true) @NotNull OTuple on(@NotNull OTuple oTuple,
    ↪ @NotNull ODouble oDouble) {
    return on(oDouble, oTuple);
}

@Contract(pure = true) @NotNull OFraction on(@NotNull OFraction
    ↪ oFraction1, @NotNull OFraction oFraction2) {
    return new OFraction(oFraction1.getFraction().add(oFraction2.
        ↪ getFraction()));
}

@Contract(pure = true) @NotNull OSet on(@NotNull OFraction oFraction,
    ↪ @NotNull OSet oSet) {
    Set<Double> newSet = new HashSet<>();
    for (double d : oSet.getSet())
        newSet.add(d + oFraction.getDouble());
    return new OSet(newSet);
}

@Contract(pure = true) @NotNull OSet on(@NotNull OSet oSet, @NotNull
    ↪ OFraction oFraction) {
    return on(oFraction, oSet);
}

@Contract(pure = true) @NotNull OMatrix on(@NotNull OFraction
    ↪ oFraction, @NotNull OMatrix oMatrix) {
    return new OMatrix(oMatrix.getMatrix().scalarAdd(oFraction.
        ↪ getDouble()));
}

@Contract(pure = true) @NotNull OMatrix on(@NotNull OMatrix oMatrix,
    ↪ @NotNull OFraction oFraction) {
    return on(oFraction, oMatrix);
}

@Contract(pure = true) @NotNull OPolynom on(@NotNull OFraction
    ↪ oFraction, @NotNull OPolynom oPolynom) {
    double[] d = oPolynom.getPolynom().getCoefficients();
    d[0] += oFraction.getDouble();
    return new OPolynom(new PolynomialFunction(d));
}

@Contract(pure = true) @NotNull OPolynom on(@NotNull OPolynom oPolynom

```

```

    ↪ , @NotNull OFraction oFraction) {
    return on(oFraction, oPolynom);
}

@Contract(pure = true) @NotNull OTuple on(@NotNull OFraction oFraction
    ↪ , @NotNull OTuple oTuple) {
    double[] oldTuple = oTuple.getTuple();
    double[] newTuple = new double[oldTuple.length];
    double fractionDouble = oFraction.getDouble();
    for (int i = 0; i < newTuple.length; i++)
        newTuple[i] = fractionDouble + oldTuple[i];
    return new OTuple(newTuple);
}

@Contract(pure = true) @NotNull OTuple on(@NotNull OTuple oTuple,
    ↪ @NotNull OFraction oFraction) {
    return on(oFraction, oTuple);
}

@Contract(pure = true) @NotNull OMatrix on(@NotNull OMatrix oMatrix1,
    ↪ @NotNull OMatrix oMatrix2) {
    return new OMatrix(oMatrix1.getMatrix().add(oMatrix2.getMatrix()));
    ↪
}

@Contract(pure = true) @NotNull OPolynom on(@NotNull OPolynom
    ↪ oPolynom1, @NotNull OPolynom oPolynom2) {
    return new OPolynom(oPolynom1.getPolynom().add(oPolynom2.
        ↪ getPolynom()));
}

@Contract(pure = true) @NotNull OTuple on(@NotNull OTuple oTuple1,
    ↪ @NotNull OTuple oTuple2) {
    double[] tuple1 = oTuple1.getTuple();
    double[] tuple2 = oTuple2.getTuple();
    double[] tupleSum = new double[tuple1.length];

    if (tuple1.length != tuple2.length)
        throw new IllegalArgumentException("Tuples must have matching
            ↪ size.");
    for (int i = 0; i < tupleSum.length; i++)
        tupleSum[i] = tuple1[i] + tuple2[i];
    return new OTuple(tupleSum);
}
}

```

Listing 25: PlusTest (Schwenke)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import org.junit.Test;

```

```

import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import de.fhdw.wip.rpntilecalculator.model.operands.OFraction;
import de.fhdw.wip.rpntilecalculator.model.operands.OMatrix;
import de.fhdw.wip.rpntilecalculator.model.operands.OPolynom;
import de.fhdw.wip.rpntilecalculator.model.operands.OSet;
import de.fhdw.wip.rpntilecalculator.model.operands OTuple;

import static org.junit.Assert.*;

/*
 * Summary: Test for the class 'PlusTest'
 * Author: Tim Schwenke
 */

public class PlusTest {

    private Plus PLUS = Plus.getInstance();

    @Test public void on_DoubleDouble_isCorrect() {
        assertTrue(PLUS.on(
            new ODouble(5),
            new ODouble(5)
        ).equalsValue(new ODouble(10)));
    }

    @Test public void on_DoubleFraction_isCorrect() {
        assertTrue(PLUS.on(
            new OFraction(2, 4),
            new OFraction(2, 4)
        ).equalsValue(new OFraction(1, 1)));
    }

    @Test public void on_FractionDouble_isCorrect() {
        assertTrue(PLUS.on(
            new OFraction(2, 2),
            new ODouble(5)
        ).equalsValue(new ODouble(6)));
    }

    @Test public void on_DoubleSet_isCorrect() {
        assertTrue(PLUS.on(
            new ODouble(1),
            new OSet(1, 2, 3)
        ).equalsValue(new OSet(2, 3, 4)));
    }

    @Test public void on_SetDouble_isCorrect() {
        assertTrue(PLUS.on(
            new OSet(1, 2, 3),
            new ODouble(1)
        ).equalsValue(new OSet(2, 3, 4)));
    }

    @Test public void on_DoubleMatrix_isCorrect() {
        assertTrue(PLUS.on(

```

```

        new ODouble(1),
        new OMatrix(new double[][] {
            {1, 2, 3},
            {1, 2, 3}
        })
    ).equalsValue(new OMatrix(new double[][] {
        {2, 3, 4},
        {2, 3, 4}
    })));
}

@Test public void on_MatrixDouble_isCorrect() {
    assertTrue(PLUS.on(
        new OMatrix(new double[][] {
            {1, 2, 3},
            {1, 2, 3}
        }),
        new ODouble(1)
    ).equalsValue(new OMatrix(new double[][] {
        {2, 3, 4},
        {2, 3, 4}
    })));
}

@Test public void on_DoublePolynom_isCorrect() {
    assertTrue(PLUS.on(
        new ODouble(1),
        new OPolynom(1, 2, 3)
    ).equalsValue(new OPolynom(2, 2, 3)));
}

@Test public void on_PolynomDouble_isCorrect() {
    assertTrue(PLUS.on(
        new OPolynom(1, 2, 3),
        new ODouble(1)
    ).equalsValue(new OPolynom(2, 2, 3)));
}

@Test public void on_DoubleTuple_isCorrect() {
    assertTrue(PLUS.on(
        new ODouble(1),
        new OTuple(1, 2, 3)
    ).equalsValue(new OTuple(2, 3, 4)));
}

@Test public void on_TupleDouble_isCorrect() {
    assertTrue(PLUS.on(
        new OTuple(1, 2, 3),
        new ODouble(1)
    ).equalsValue(new OTuple(2, 3, 4)));
}

@Test public void on_FractionFraction_isCorrect() {
    assertTrue(PLUS.on(

```

```

        new OFraction(1, 1),
        new OFraction(1, 1)
    ).equalsValue(new OFraction(2, 1)));
}

@Test public void on_FractionSet_isCorrect() {
    assertTrue(PLUS.on(
        new OFraction(1, 1),
        new OSet(1, 2, 3)
    ).equalsValue(new OSet(2, 3, 4)));
}

@Test public void on_SetFraction_isCorrect() {
    assertTrue(PLUS.on(
        new OSet(1, 2, 3),
        new OFraction(1, 1)
    ).equalsValue(new OSet(2, 3, 4)));
}

@Test public void on_FractionMatrix_isCorrect() {
    assertTrue(PLUS.on(
        new OFraction(1, 1),
        new OMatrix(new double[][]{
            {1, 2, 3},
            {1, 2, 3}
        })
    ).equalsValue(new OMatrix(new double[][]{
        {2, 3, 4},
        {2, 3, 4}
    })));
}

@Test public void on_MatrixFraction_isCorrect() {
    assertTrue(PLUS.on(
        new OMatrix(new double[][]{
            {1, 2, 3},
            {1, 2, 3}
        }),
        new OFraction(1, 1)
    ).equalsValue(new OMatrix(new double[][]{
        {2, 3, 4},
        {2, 3, 4}
    })));
}

@Test public void on_FractionPolynom_isCorrect() {
    assertTrue(PLUS.on(
        new OFraction(1, 1),
        new OPolynom(1, 2, 3)
    ).equalsValue(new OPolynom(2, 2, 3)));
}

@Test public void on_PolynomFraction_isCorrect() {
    assertTrue(PLUS.on(

```

```

        new OPolynom(1, 2, 3),
        new OFraction(1, 1)
    ).equalsValue(new OPolynom(2, 2, 3)));
}

@Test public void on_FractionTuple_isCorrect() {
    assertTrue(PLUS.on(
        new OFraction(1, 1),
        new OTuple(1, 2, 3)
    ).equalsValue(new OTuple(2, 3, 4)));
}

@Test public void on_TupleFraction_isCorrect() {
    assertTrue(PLUS.on(
        new OTuple(1, 2, 3),
        new OFraction(1, 1)
    ).equalsValue(new OTuple(2, 3, 4)));
}

@Test public void on_MatrixMatrix_isCorrect() {
    assertTrue(PLUS.on(
        new OMatrix(new double[][]{
            {1, 2, 3},
            {1, 2, 3}
        }),
        new OMatrix(new double[][]{
            {1, 2, 3},
            {1, 2, 3}
        })
    ).equalsValue(new OMatrix(new double[][]{
        {2, 4, 6},
        {2, 4, 6}
    })));
}

@Test public void on_MatrixMatrix_isWrongDimension() {
    try {
        PLUS.on(
            new OMatrix(new double[][]{
                {1, 2, 3},
                {1, 2, 3}
            }),
            new OMatrix(new double[][]{
                {1, 2, 3}
            })
        );
    } catch (RuntimeException e) {
        assertTrue("Should always fail", true);
    }
}

@Test public void on_PolynomPolynom_isCorrect() {
    assertTrue(PLUS.on(
        new OPolynom(1, 2, 3),

```

```

        new OPolynom(0, 0, 3, 4, 5)
    ).equalsValue(new OPolynom(1, 2, 6, 4 ,5)));
}

@Test public void on_TupleTuple_isCorrect() {
    assertTrue(PLUS.on(
        new OTuple(1, 2, 3),
        new OTuple(1, 2, 3)
    ).equalsValue(new OTuple(2, 4, 6)))
};

@Test public void on_TupleTuple_isWrongDimension() {
    try {
        PLUS.on(
            new OTuple(1, 2, 3),
            new OTuple(1, 2, 3, 4)
        );
        fail();
    } catch (RuntimeException e) {
        assertTrue("Should always fail", true);
    }
}
}

```

Listing 26: Power (Falk)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import de.fhdw.wip.rpntilecalculator.model.operands.OFraction;
import de.fhdw.wip.rpntilecalculator.model.operands.OMatrix;
import de.fhdw.wip.rpntilecalculator.model.operands.OTuple;
import de.fhdw.wip.rpntilecalculator.model.operands.Operand;

import java.lang.Math;
import java.lang.reflect.Array;

import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Contract;

/*
 * Summary: Defines the Power click.
 * Author: Hendrik Falk
 */

public class Power extends Action{

    @NotNull private static final Power POWER = new Power();
    @NotNull private static final Times TIMES = Times.getInstance();

    @Contract(pure = true) @NotNull public static Power getInstance() {
        return POWER; }
    private Power() {
        requiredNumOfOperands = new int[]{2};
    }
}

```

```

}

@NotNull @Override
public Operand with(@NotNull Operand... operands) throws
    ↪ CalculationException {
    scopedAction = this;
    return super.with(operands);
}

//region Double
//
    ↪ -----
    ↪

@Contract(pure = true) @NotNull ODouble on(@NotNull ODouble base,
    ↪ @NotNull ODouble exponent) {
    return new ODouble(Math.pow(base.getDouble(), exponent.getDouble())
        ↪ );
}

@Contract(pure = true) @NotNull ODouble on(@NotNull ODouble base,
    ↪ @NotNull OFraction exponent) {
    return new ODouble(Math.pow(base.getDouble(), exponent.getDouble())
        ↪ );
}

//region Fraction
//
    ↪ -----
    ↪

@Contract(pure = true) @NotNull OFraction on(@NotNull OFraction base,
    ↪ @NotNull ODouble exponent){
    return new OFraction(Math.pow(base.getDouble(), exponent.getDouble()
        ↪ ()));
}

@Contract(pure = true) @NotNull OFraction on(@NotNull OFraction base,
    ↪ @NotNull OFraction exponent){
    return new OFraction(Math.pow(base.getDouble(), exponent.getDouble()
        ↪ ()));
}

//region Matrix
//
    ↪ -----
    ↪

@Contract(pure = true) @NotNull OMatrix on(@NotNull OMatrix base,
    ↪ @NotNull ODouble exponent) {
    if(base.getMatrix().isSquare()) {
        OMatrix resultMatrix = TIMES.on(base, base);

        if (exponent.getDouble() > 2) {
            for (int i = 2; i < exponent.getDouble(); i++)
}

```

```

        resultMatrix = TIMES.on(resultMatrix, base);
    }
    return resultMatrix;
}else
{
    throw new IllegalArgumentException("You need a square matrix
        ↪ for power operation.");
}
}

@Contract(pure = true) @NotNull OMatrix on(@NotNull OMatrix base,
    ↪ @NotNull OFraction exponent){
    return TIMES.on(TIMES.on(exponent, base), base);
}

//region Vector
//
    ↪ -----
    ↪

@Contract(pure = true) @NotNull OTuple on(@NotNull OTuple base,
    ↪ @NotNull ODouble exponent) {
    double[] arrayTuple = base.getTuple();
    if(arrayTuple.length == 1)
    {
        arrayTuple[0] = Math.pow(Array.getDouble(arrayTuple,0),
            ↪ exponent.getDouble());
        return new OTuple(arrayTuple);
    }else
    {
        throw new IllegalArgumentException("You need a square matrix
            ↪ for power operation.");
    }
}

@Contract(pure = true) @NotNull OTuple on(@NotNull OTuple base,
    ↪ @NotNull OFraction exponent) {
    return on(base, new ODouble(exponent.getDouble()));
}
}

```

Listing 27: Sinus (Keienburg)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import de.fhdw.wip.rpntilecalculator.model.operands.Operand;

import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

/*
 * Summary: Defines the Sinus action.
 * Author: Jannis Luca Keienburg
 */

```

```

public class Sinus extends Action {

    @NotNull
    private static final Sinus SINUS = new Sinus();

    /*
     * Singleton for SINUS
     * @return singleton object
     */
    @Contract(pure = true) @NotNull public static Sinus getInstance() {
        ↪ return SINUS; }

    private Sinus() {
        requiredNumOfOperands = new int[]{1, 2};
    }

    @NotNull @Override
    public Operand with(@NotNull Operand... operands) throws
        ↪ CalculationException {
        scopedAction = this;
        return super.with(operands);
    }

    // Calculates the sinus with a given angle.
    @Contract(pure = true) @NotNull ODouble on(@NotNull ODouble angle) {
        return new ODouble(Math.sin(Math.toRadians((angle.getDouble()))));
    }
}

```

Listing 28: Slash (Falk)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import de.fhdw.wip.rpntilecalculator.model.operands.DoubleComparator;
import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import de.fhdw.wip.rpntilecalculator.model.operands.OFraction;
import de.fhdw.wip.rpntilecalculator.model.operands.OMatrix;
import de.fhdw.wip.rpntilecalculator.model.operands.OPolynom;
import de.fhdw.wip.rpntilecalculator.model.operands.OSet;
import de.fhdw.wip.rpntilecalculator.model.operands OTuple;
import de.fhdw.wip.rpntilecalculator.model.operands.Operand;

import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

/*
 * Summary: Defines the Slash action.
 * Author: Hendrik Falk
 */
public class Slash extends Action {

    @NotNull private static final Times TIMES = Times.getInstance();
    @NotNull private static final Slash SLASH = new Slash();

    @Contract(pure = true) @NotNull public static Slash getInstance() {
        ↪ return SLASH; }

    private Slash() {

```

```

        requiredNumOfOperands = new int[]{2};
    }

    @NotNull @Override
    public Operand with(@NotNull Operand... operands) throws
        ↪ CalculationException {
        scopedAction = this;
        return super.with(operands);
    }

    //region Double
    //
    ↪ -----
    ↪

    @Contract(pure = true) @NotNull ODouble on(@NotNull ODouble oDouble1,
        ↪ @NotNull ODouble oDouble2) {
        if (DoubleComparator.isZero(oDouble2.getDouble()))
            throw new IllegalArgumentException("Division by Zero not
                ↪ allowed");
        return new ODouble(oDouble1.getDouble() / oDouble2.getDouble());
    }

    @Contract(pure = true) @NotNull ODouble on(@NotNull ODouble oDouble,
        ↪ @NotNull OFraction oFraction) {
        if (DoubleComparator.isZero(oFraction.getDouble()))
            throw new IllegalArgumentException("Division by Zero not
                ↪ allowed");
        return new ODouble(oDouble.getDouble() / oFraction.getDouble());
    }

    //endregion

    //region Fraction
    //
    ↪ -----
    ↪

    @Contract(pure = true) @NotNull OFraction on(@NotNull OFraction
        ↪ oFraction1, @NotNull OFraction oFraction2) {
        if (DoubleComparator.isZero(oFraction2.getDouble()))
            throw new IllegalArgumentException("Division by Zero not
                ↪ allowed");
        return new OFraction(oFraction1.getFraction().divide(oFraction2.
            ↪ getFraction()));
    }

    @Contract(pure = true) @NotNull ODouble on(@NotNull OFraction
        ↪ oFraction, @NotNull ODouble oDouble) {
        if (DoubleComparator.isZero(oDouble.getDouble()))
            throw new IllegalArgumentException("Division by Zero not
                ↪ allowed");
        return new ODouble(oFraction.getDouble() / oDouble.getDouble());
    }
}

```

```

//endregion

//region Set
//
//      ↪ -----
//      ↪

@Contract(pure = true) @NotNull OSet on(@NotNull OSet oSet, @NotNull
    ↪ ODouble oDouble) {
    if (DoubleComparator.isZero(oDouble.getDouble()))
        throw new IllegalArgumentException("Division by Zero not
            ↪ allowed");
    return TIMES.on(oDouble.inverseValue(), oSet);
}

@Contract(pure = true) @NotNull OSet on(@NotNull OSet oSet, @NotNull
    ↪ OFraction oFraction) {
    if (DoubleComparator.isZero(oFraction.getDouble()))
        throw new IllegalArgumentException("Division by Zero not
            ↪ allowed");
    return on(oSet, new ODouble(oFraction.getDouble()));
}

//endregion

//region Matrix
//
//      ↪ -----
//      ↪

@Contract(pure = true) @NotNull OMatrix on(@NotNull OMatrix oMatrix,
    ↪ @NotNull ODouble oDouble) {
    if (DoubleComparator.isZero(oDouble.getDouble()))
        throw new IllegalArgumentException("Division by Zero not
            ↪ allowed");
    return TIMES.on(oDouble.inverseValue(), oMatrix);
}

@Contract(pure = true) @NotNull OMatrix on(@NotNull OMatrix oMatrix,
    ↪ @NotNull OFraction oFraction) {
    if (DoubleComparator.isZero(oFraction.getDouble()))
        throw new IllegalArgumentException("Division by Zero not
            ↪ allowed");
    return on(oMatrix, new ODouble(oFraction.getDouble()));
}

//endregion

//region Polynom
//
//      ↪ -----
//      ↪

@Contract(pure = true) @NotNull OPolynom on(@NotNull OPolynom

```

```

    ↵ oPolynom1, @NotNull OPolynom oPolynom2) {
    for (double d : oPolynom2.getPolynom().getCoefficients())
        if (DoubleComparator.isZero(d))
            throw new IllegalArgumentException("Division by Zero not
                ↵ allowed");
    return new OPolynom(oPolynom1.getPolynom().multiply(oPolynom2.
        ↵ inverseValue().getPolynom()));
}

@Contract(pure = true) @NotNull OPolynom on(@NotNull OPolynom oPolynom
    ↵ , @NotNull ODouble oDouble) {
    if (DoubleComparator.isZero(oDouble.getDouble()))
        throw new IllegalArgumentException("Division by Zero not
            ↵ allowed");
    return TIMES.on(oDouble.inverseValue(), oPolynom);
}

@Contract(pure = true) @NotNull OPolynom on(@NotNull OPolynom oPolynom
    ↵ , @NotNull OFraction oFraction) {
    if (DoubleComparator.isZero(oFraction.getDouble()))
        throw new IllegalArgumentException("Division by Zero not
            ↵ allowed");
    return on(oPolynom, new ODouble(oFraction.getDouble()));
}

//endregion

//region Tuple
//
    ↵ -----
    ↵

@Contract(pure = true) @NotNull OTuple on(@NotNull OTuple oTuple1,
    ↵ @NotNull OTuple oTuple2) {
    for (double d : oTuple2.getTuple())
        if (DoubleComparator.isZero(d))
            throw new IllegalArgumentException("Division by Zero not
                ↵ allowed");
    return TIMES.on(oTuple1, oTuple2.inverseValue());
}

@Contract(pure = true) @NotNull OTuple on(@NotNull OTuple oTuple,
    ↵ @NotNull ODouble oDouble) {
    if (DoubleComparator.isZero(oDouble.getDouble()))
        throw new IllegalArgumentException("Division by Zero not
            ↵ allowed");
    return TIMES.on(oDouble.inverseValue(), oTuple);
}

@Contract(pure = true) @NotNull OTuple on(@NotNull OTuple oTuple,
    ↵ @NotNull OFraction oFraction) {
    if (DoubleComparator.isZero(oFraction.getDouble()))
        throw new IllegalArgumentException("Division by Zero not
            ↵ allowed");
}

```

```

        return TIMES.on(oFraction.inverseValue(), oTuple);
    }

    //endregion
}

```

Listing 29: SlashTest (Schwenke)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import org.junit.Test;

import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import de.fhdw.wip.rpntilecalculator.model.operands.OFraction;
import de.fhdw.wip.rpntilecalculator.model.operands.OMatrix;
import de.fhdw.wip.rpntilecalculator.model.operands.OPolynom;
import de.fhdw.wip.rpntilecalculator.model.operands.OSet;
import de.fhdw.wip.rpntilecalculator.model.operands.OTuple;

import static org.junit.Assert.*;

/*
 * Summary: Test for the class 'SlashTest'
 * Author: Tim Schwenke
 */

public class SlashTest {

    private Slash SLASH = Slash.getInstance();

    @Test public void on_DoubleDouble_isCorrect() {
        assertTrue(SLASH.on(
            new ODouble(4),
            new ODouble(2)
        ).equalsValue(new ODouble(2)));
    }

    @Test public void on_DoubleDouble_isDividedByZero() {
        try {
            SLASH.on(
                new ODouble(4),
                new ODouble(0)
            );
            fail();
        } catch (RuntimeException e) {
            assertTrue(true);
        }
    }

    @Test public void on_DoubleFraction_isCorrect() {
        assertTrue(SLASH.on(
            new ODouble(10),
            new OFraction(1, 2)
        ).equalsValue(new ODouble(20)));
    }
}

```

```
@Test public void on_DoubleFraction_isDividedByZero() {
    try {
        SLASH.on(
            new ODouble(4),
            new OFraction(5, 0)
        );
        fail();
    } catch (RuntimeException e) {
        assertTrue(true);
    }
}

@Test public void on_FractionFraction_isCorrect() {
    assertTrue(SLASH.on(
        new OFraction(1, 2),
        new OFraction(1, 2)
    ).equalsValue(new OFraction(1, 1)));
}

@Test public void on_FractionDouble_isCorrect() {
    assertTrue(SLASH.on(
        new OFraction(1, 1),
        new ODouble(0.5)
    ).equalsValue(new ODouble(2)));
}

@Test public void on_FractionDouble_isDividedByZero() {
    try {
        SLASH.on(
            new OFraction(1, 1),
            new ODouble(0)
        );
        fail();
    } catch (RuntimeException e) {
        assertTrue(true);
    }
}

@Test public void on_SetDouble_isCorrect() {
    assertTrue(SLASH.on(
        new OSet(1, 2),
        new ODouble(2)
    ).equalsValue(new OSet(0.5, 1)));
}

@Test public void on_SetFraction_isCorrect() {
    assertTrue(SLASH.on(
        new OSet(1, 2),
        new OFraction(1, 2)
    ).equalsValue(new OSet(2, 4)));
}

@Test public void on_MatrixDouble_isCorrect() {
```

```

        assertTrue(SLASH.on(
            new OMatrix(new double[][] {
                {1, 2},
                {1, 2}
            }),
            new ODouble(2)
        ).equalsValue(new OMatrix(new double[][] {
            {0.5, 1},
            {0.5, 1}
        })));
    }

    @Test public void on_MatrixDouble_isDividedByZero() {
        try {
            SLASH.on(
                new OMatrix(new double[][] {
                    {1, 2},
                    {1, 2}
                }),
                new ODouble(0)
            );
            fail();
        } catch (RuntimeException e) {
            assertTrue(true);
        }
    }

    @Test public void on_MatrixFraction_isCorrect() {
        assertTrue(SLASH.on(
            new OMatrix(new double[][] {
                {1, 2},
                {1, 2}
            }),
            new OFraction(2, 1)
        ).equalsValue(new OMatrix(new double[][] {
            {0.5, 1},
            {0.5, 1}
        })));
    }

    @Test public void on_PolynomPolynom_isCorrect() {
        assertTrue(SLASH.on(
            new OPolynom(1, 2, 3),
            new OPolynom(2, 2, 2, 2)
        ).equalsValue(new OPolynom(0.5, 1.5, 3, 3, 2.5, 1.5)));
    }

    @Test public void on_PolynomPolynom_isDividedByZero() {
        try {
            SLASH.on(
                new OPolynom(1, 2, 3),
                new OPolynom(0, 2, 0, 2)
            );
            fail();
        }
    }
}

```

```

        } catch (RuntimeException e) {
            assertTrue(true);
        }
    }

@Test public void on_PolynomDouble_isCorrect() {
    assertTrue(SLASH.on(
        new OPolynom(1, 2),
        new ODouble(1)
    ).equalsValue(new OPolynom(1, 2)));
}

@Test public void on_PolynomFraction_isCorrect() {
    assertTrue(SLASH.on(
        new OPolynom(1, 2),
        new OFraction(1)
    ).equalsValue(new OPolynom(1, 2)));
}

@Test public void on_TupleTuple_isCorrect() {
    assertTrue(SLASH.on(
        new OTuple(1, 2),
        new OTuple(1, 2)
    ).equalsValue(new OTuple(1, 1)));
}

@Test public void on_TupleDouble_isCorrect() {
    assertTrue(SLASH.on(
        new OTuple(1, 2),
        new OTuple(2, 2)
    ).equalsValue(new OTuple(0.5, 1)));
}

@Test public void on_TupleFraction_isCorrect() {
    assertTrue(SLASH.on(
        new OTuple(1, 2),
        new OFraction(1, 1)
    ).equalsValue(new OTuple(1, 2)));
}
}

```

Listing 30: Tangens (Keienburg)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import de.fhdw.wip.rpntilecalculator.model.operands.Operand;

import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

/*
 * Summary: Defines the Tangens action.
 * Author: Jannis Luca Keienburg
 */

```

```

public class Tangens extends Action {

    @NotNull
    private static final Tangens TANGENS = new Tangens();

    /*
     * Singleton for SINUS
     * @return singleton object
     */
    @Contract(pure = true) @NotNull public static Tangens getInstance() {
        ↪ return TANGENS; }

    private Tangens() {
        requiredNumOfOperands = new int[]{1,2};
    }

    @NotNull @Override
    public Operand with(@NotNull Operand... operands) throws
        ↪ CalculationException {
        scopedAction = this;
        return super.with(operands);
    }

    // Calculates the tangens with a given angle.
    @Contract(pure = true) @NotNull ODouble on(@NotNull ODouble angle) {
        return new ODouble(Math.tan(Math.toRadians((angle.getDouble()))));
    }
}

```

Listing 31: Times (Falk)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import de.fhdw.wip.rpntilecalculator.model.operands.OFraction;
import de.fhdw.wip.rpntilecalculator.model.operands.OMatrix;
import de.fhdw.wip.rpntilecalculator.model.operands.OPolynom;
import de.fhdw.wip.rpntilecalculator.model.operands.OSet;
import de.fhdw.wip.rpntilecalculator.model.operands.OTuple;
import de.fhdw.wip.rpntilecalculator.model.operands.Operand;

import org.apache.commons.math3.analysis.polynomials.PolynomialFunction;
import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

import java.util.HashSet;
import java.util.Set;

/*
 * Summary: Defines the Times click. Lets the user Multiplies operands.
 * Author: Hendrik
 */
@SuppressWarnings({"unused"})
public class Times extends Action {

    @NotNull private static final Times TIMES = new Times();

```

```

/*
 * Singleton for TIMES
 * @return singleton object
 */
@Contract(pure = true) @NotNull public static Times getInstance() {
    ↪ return TIMES; }
private Times() {
    requiredNumOfOperands = new int[]{2};
}

/*
 * Multiplying ODouble and ODouble
 * @param operands params
 * @return product of operands
 */
@NotNull @Override
public Operand with(@NotNull Operand... operands) throws
    ↪ CalculationException {
    scopedAction = this;
    return super.with(operands);
}

//region Double
//
//  
-----  

//  
-----  

/*
 * Multiplying ODouble and ODouble
 * @param oDouble1 first operand
 * @param oDouble2 second operand
 * @return product of params
 */
@Contract(pure = true) @NotNull ODouble on(@NotNull ODouble oDouble1,
    ↪ @NotNull ODouble oDouble2) {
    return new ODouble(oDouble1.getDouble() * oDouble2.getDouble());
}

/*
 * Multiplying ODouble and OFraction
 * @param oDouble first operand
 * @param oFraction second operand
 * @return product of params
 */
@Contract(pure = true) @NotNull ODouble on(@NotNull ODouble oDouble,
    ↪ @NotNull OFraction oFraction) {
    return new ODouble(oDouble.getDouble() * oFraction.getDouble());
}

@Contract(pure = true) @NotNull ODouble on(@NotNull OFraction
    ↪ oFraction, @NotNull ODouble oDouble) {
    return on(oDouble, oFraction);
}

// endregion

```

```

//region OSet
//
/* 
 * Multiplying ODouble and oSet
 * @param oDouble first operand
 * @param oSet second operand
 * @return product of params
 */
@Contract(pure = true) @NotNull OSet on(@NotNull ODouble oDouble,
    ↪ @NotNull OSet oSet) {
    Set<Double> newSet = new HashSet<>();
    for (double d : oSet.getSet())
        newSet.add(d * oDouble.getDouble());
    return new OSet(newSet);
}

@Contract(pure = true) @NotNull OSet on(@NotNull OSet oSet, @NotNull
    ↪ ODouble oDouble) {
    return on(oDouble, oSet);
}

/*
 * Multiplying ODouble and oMatrix
 * @param oDouble first operand
 * @param oMatrix second operand
 * @return product of params
 */
@Contract(pure = true) @NotNull OMatrix on(@NotNull ODouble oDouble,
    ↪ @NotNull OMatrix oMatrix) {
    return new OMatrix(oMatrix.getMatrix().scalarMultiply(oDouble.
        ↪ getDouble()));
}

@Contract(pure = true) @NotNull OMatrix on(@NotNull OMatrix oMatrix,
    ↪ @NotNull ODouble oDouble) {
    return on(oDouble, oMatrix);
}

/*
 * Multiplying ODouble and oPolynom
 * @param oDouble first operand
 * @param oPolynom second operand
 * @return product of params
 */
@Contract(pure = true) @NotNull OPolynom on(@NotNull ODouble oDouble,
    ↪ @NotNull OPolynom oPolynom) {
    double[] d = oPolynom.getPolynom().getCoefficients();
    for (int i = 0; i < d.length; i++)
        d[i] *= oDouble.getDouble();
    return new OPolynom(new PolynomialFunction(d));
}

```

```

@Contract(pure = true) @NotNull OPolynom on(@NotNull OPolynom oPolynom
    ↪ , @NotNull ODouble oDouble) {
    return on(oDouble, oPolynom);
}

/*
 * Multiplying ODouble and oTuple
 * @param oDouble first operand
 * @param oTuple second operand
 * @return product of params
 */
@Contract(pure = true) @NotNull OTuple on(@NotNull ODouble oDouble,
    ↪ @NotNull OTuple oTuple) {
    double[] oldTuple = oTuple.getTuple();
    double[] newTuple = new double[oldTuple.length];
    for (int i = 0; i < newTuple.length; i++)
        newTuple[i] = oldTuple[i] * oDouble.getDouble();
    return new OTuple(newTuple);
}

@Contract(pure = true) @NotNull OTuple on(@NotNull OTuple oTuple,
    ↪ @NotNull ODouble oDouble) {
    return on(oDouble, oTuple);
}

//endregion

//region Fraction
//  

    ↪ -----  

    ↪

/*
 * Multiplying OFraction and OFraction
 * @param oFraction1 first operand
 * @param oFraction2 second operand
 * @return product of params
 */
@Contract(pure = true) @NotNull OFraction on(@NotNull OFraction
    ↪ oFraction1, @NotNull OFraction oFraction2) {
    return new OFraction(oFraction1.getFraction().multiply(oFraction2.
        ↪ getFraction()));
}

/*
 * Multiplying OFraction and OSet
 * @param oFraction first operand
 * @param oSet second operand
 * @return product of params
 */
@Contract(pure = true) @NotNull OSet on(@NotNull OFraction oFraction,
    ↪ @NotNull OSet oSet) {
    return on(new ODouble(oFraction.getDouble()), oSet);
}

```

```

/*
 * Multiplying OFraction and OMatrix
 * @param oFraction first operand
 * @param oMatrix second operand
 * @return product of params
 */
@Contract(pure = true) @NotNull OMatrix on(@NotNull OFraction
    ↪ oFraction, @NotNull OMatrix oMatrix) {
    return new OMatrix(oMatrix.getMatrix().scalarMultiply(oFraction.
        ↪ getDouble()));
}

@Contract(pure = true) @NotNull OMatrix on(@NotNull OMatrix oMatrix,
    ↪ @NotNull OFraction oFraction) {
    return on(oFraction, oMatrix);
}

/*
 * Multiplying OFraction and oPolynom
 * @param oFraction first operand
 * @param oPolynom second operand
 * @return product of params
 */
@Contract(pure = true) @NotNull OPolynom on(@NotNull OFraction
    ↪ oFraction, @NotNull OPolynom oPolynom) {
    return on(new ODouble(oFraction.getDouble()), oPolynom);
}

/*
 * Multiplying OFraction and oTuple
 * @param oFraction first operand
 * @param oTuple second operand
 * @return product of params
 */
@Contract(pure = true) @NotNull OTuple on(@NotNull OFraction oFraction
    ↪ , @NotNull OTuple oTuple) {
    return on(new ODouble(oFraction.getDouble()), oTuple);
}

//endregion

/*
 * Multiplying OMatrix and OMatrix
 * @param oMatrix1 first operand
 * @param oMatrix2 second operand
 * @return product of params
 */
@Contract(pure = true) @NotNull OMatrix on(@NotNull OMatrix oMatrix1,
    ↪ @NotNull OMatrix oMatrix2) {
    return new OMatrix(oMatrix1.getMatrix().multiply(oMatrix2.
        ↪ getMatrix()));
}

/*

```

```

    * Multiplying OPolynom and OPolynom
    * @param oPolynom1 first operand
    * @param oPolynom2 second operand
    * @return product of params
*/
@Contract(pure = true) @NotNull OPolynom on(@NotNull OPolynom
    ↪ oPolynom1, @NotNull OPolynom oPolynom2) {
    return new OPolynom(oPolynom1.getPolynom().multiply(oPolynom2.
        ↪ getPolynom()));
}

/*
    * Multiplying OTuple and OTuple
    * @param oTuple1 first operand
    * @param oTuple2 second operand
    * @return product of params
*/
@Contract(pure = true) @NotNull OTuple on(@NotNull OTuple oTuple1,
    ↪ @NotNull OTuple oTuple2) {
    double[] tuple1 = oTuple1.getTuple();
    double[] tuple2 = oTuple2.getTuple();
    double[] tupleSum = new double[tuple2.length];

    if (tuple1.length != tuple2.length)
        throw new IllegalArgumentException("Tuples must have matching
            ↪ size.");
    for (int i = 0; i < tuple1.length; i++)
        tupleSum[i] = tuple1[i] * tuple2[i];
    return new OTuple(tupleSum);
}
}

```

Listing 32: TimesTest (Schwenke)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import org.junit.Test;

import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import de.fhdw.wip.rpntilecalculator.model.operands.OFraction;
import de.fhdw.wip.rpntilecalculator.model.operands.OMatrix;
import de.fhdw.wip.rpntilecalculator.model.operands.OPolynom;
import de.fhdw.wip.rpntilecalculator.model.operands.OSet;
import de.fhdw.wip.rpntilecalculator.model.operands.OTuple;

import static org.junit.Assert.*;

/*
 * Summary: Test for the class 'TimesTest'
 * Author: Tim Schwenke
 */

public class TimesTest {

```

```
private Times TIMES = Times.getInstance();

@Test public void on_DoubleDouble_isCorrect() {
    assertTrue(TIMES.on(
        new ODouble(5),
        new ODouble(5)
    ).equalsValue(new ODouble(25)));
}

@Test public void on_DoubleFraction_isCorrect() {
    assertTrue(TIMES.on(
        new ODouble(5),
        new OFraction(5, 1)
    ).equalsValue(new ODouble(25)));
}

@Test public void on_FractionDouble_isCorrect() {
    assertTrue(TIMES.on(
        new OFraction(1, 1),
        new ODouble(5)
    ).equalsValue(new ODouble(5)));
}

@Test public void on_DoubleSet_isCorrect() {
    assertTrue(TIMES.on(
        new ODouble(2),
        new OSet(1, 2)
    ).equalsValue(new OSet(2, 4)));
}

@Test public void on_SetDouble_isCorrect() {
    assertTrue(TIMES.on(
        new OSet(1, 2),
        new ODouble(2)
    ).equalsValue(new OSet(2, 4)));
}

@Test public void on_DoubleMatrix_isCorrect() {
    assertTrue(TIMES.on(
        new ODouble(2),
        new OMatrix(new double[][]{{1, 2}})
    ).equalsValue(new OMatrix(new double[][]{{2, 4}})));
}

@Test public void on_MatrixDouble_isCorrect() {
    assertTrue(TIMES.on(
        new OMatrix(new double[][]{{1, 2}}),
        new ODouble(2)
    ).equalsValue(new OMatrix(new double[][]{{2, 4}})));
}

@Test public void on_DoublePolynom_isCorrect() {
    assertTrue(TIMES.on(
```

```

        new ODouble(2),
        new OPolynom(0, 1, 2)
    ).equalsValue(new OPolynom(0, 2, 4)));
}

@Test public void on_PolynomDouble_isCorrect() {
    assertTrue(TIMES.on(
        new OPolynom(0, 1, 2),
        new ODouble(2)
    ).equalsValue(new OPolynom(0, 2, 4)));
}

@Test public void on_DoubleTuple_isCorrect() {
    assertTrue(TIMES.on(
        new ODouble(2),
        new OTuple(1, 2)
    ).equalsValue(new OTuple(2, 4)));
}

@Test public void on_TupleDouble_isCorrect() {
    assertTrue(TIMES.on(
        new OTuple(1, 2),
        new ODouble(2)
    ).equalsValue(new OTuple(2, 4)));
}

@Test public void on_FractionFraction_isCorrect() {
    assertTrue(TIMES.on(
        new OFraction(2, 1),
        new OFraction(2, 1)
    ).equalsValue(new OFraction(4, 1)));
}

@Test public void on_FractionSet_isCorrect() {
    assertTrue(TIMES.on(
        new OFraction(1, 1),
        new OSet(1, 2)
    ).equalsValue(new OSet(1, 2)));
}

@Test public void on_FractionMatrix_isCorrect() {
    assertTrue(TIMES.on(
        new OFraction(1, 1),
        new OMatrix(new double[][] {
            {1, 2},
            {1, 2}
        })
    ).equalsValue(new OMatrix(new double[][] {
        {1, 2},
        {1, 2}
    })));
}

@Test public void on_FractionPolynom_isCorrect() {

```

```

        assertTrue(TIMES.on(
            new OFraction(1, 1),
            new OPolynom(0, 1, 3)
        ).equalsValue(new OPolynom(0, 1, 3)));
    }

    @Test public void on_FractionTuple_isCorrect() {
        assertTrue(TIMES.on(
            new OFraction(1, 1),
            new OTuple(1, 2)
        ).equalsValue(new OTuple(1, 2)));
    }

    @Test public void on_MatrixMatrix_isCorrect() {
        assertTrue(TIMES.on(
            new OMatrix(new double[][]{
                {1, 2},
                {1, 2}
            }),
            new OMatrix(new double[][]{
                {1, 2},
                {1, 2}
            })
        ).equalsValue(new OMatrix(new double[][]{
            {3, 6},
            {3, 6}
        })));
    }

    @Test public void on_PolynomPolynom_isCorrect() {
        assertTrue(TIMES.on(
            new OPolynom(1, 2),
            new OPolynom(2, 3)
        ).equalsValue(new OPolynom(2, 7, 6)));
    }

    @Test public void on_TupleTuple_isCorrect() {
        assertTrue(TIMES.on(
            new OTuple(1, 2),
            new OTuple(1, 2)
        ).equalsValue(new OTuple(1, 4)));
    }
}

```

Listing 33: Zeros (Keienburg)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import de.fhdw.wip.rpntilecalculator.model.operands.OPolynom;
import de.fhdw.wip.rpntilecalculator.model.operands.OTuple;
import de.fhdw.wip.rpntilecalculator.model.operands.Operand;

```

```

/*
 * Summary: A Class that can calculate the zeros of functions and
 *           ↪ quadratic functions.
 * Author: Jannis Luca Keienburg
 */
public class Zeros extends Action {

    @NotNull private static final Zeros ZEROS = new Zeros();

    @Contract(pure = true) @NotNull public static Zeros getInstance() {
        ↪ return ZEROS; }
    private Zeros() {requiredNumOfOperands = new int[] {1}; }

    @NotNull @Override
    public Operand with(@NotNull Operand... operands) throws
        ↪ CalculationException {
        scopedAction = this;
        return super.with(operands);
    }

    @Contract(pure = true) @NotNull OTuple on(@NotNull OPolynom oPolynom)
        ↪ {
        double[] results = calculateZeros(oPolynom);
        return new OTuple(results);
    }

    // Function that calculates the zeros when called
    public double [] calculateZeros(OPolynom oPolynom)
    {
        double [] zeros = new double[] {};
        double[] functionAsDouble = oPolynom.getPolynom().getCoefficients()
            ↪ ;
        int type = normalOrQuadraticFunction(functionAsDouble);
        if (type == 1)
        {
            zeros = zerosTypeOne(functionAsDouble);
        }
        else if(type == 2)
        {
            zeros = zerosTypeTwo(functionAsDouble);
        }
        return zeros;
    }

    // Checks if it is a normal function or an quadratic function
    private int normalOrQuadraticFunction(double [] functionAsDouble)
    {
        int result = 1;
        if(functionAsDouble.length == 3)
        {
            result = 2;
        }
        return result;
    }
}

```

```

// Calculates the zeros for functions like:
// a * x + b = 0
private double[] zerosTypeOne(double [] functionAsDouble)
{
    double [] zeros = new double[1];
    zeros[0] = (functionAsDouble[1] * (-1)) / functionAsDouble[0];
    return zeros;
}

// Calculates the zeros for functions like:
// a * x^2 + b * x + c = 0
// uses the Mitternachtsformel
private double[] zerosTypeTwo(double [] functionAsDouble)
{
    double [] zeros = new double[2];
    zeros[0] = -functionAsDouble[1]
        - Math.sqrt(((functionAsDouble[1] * functionAsDouble[1]) -
                    ↪ (4 * functionAsDouble[0] * functionAsDouble[2])) )
        / (2 * functionAsDouble[0]);
    zeros[1] = -functionAsDouble[1]
        + Math.sqrt(((functionAsDouble[1] * functionAsDouble[1]) -
                    ↪ (4 * functionAsDouble[0] * functionAsDouble[2])) )
        / (2 * functionAsDouble[0]);
    return zeros;
}
}

```

Listing 34: Root (Falk)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import de.fhdw.wip.rpntilecalculator.model.operands.OFraction;
import de.fhdw.wip.rpntilecalculator.model.operands.OMatrix;
import de.fhdw.wip.rpntilecalculator.model.operands.Operand;

import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Contract;

/*
 * Summary: Defines the Root click.
 * Author: Hendrik Falk
 */

public class Root extends Action{
    @NotNull private static final Root ROOT = new Root();
    @NotNull private static final Power POWER = Power.getInstance();

    @Contract(pure = true) @NotNull public static Root getInstance() {
        ↪ return ROOT; }

    private Root() {
        requiredNumOfOperands = new int[]{2};
    }
}

```

```

    @NotNull @Override
    public Operand with(@NotNull Operand... operands) throws
        CalculationException {
        scopedAction = this;
        return super.with(operands);
    }

    //region Double
    //
    // -->
    // -->
    //

    @Contract(pure = true) @NotNull ODouble on(@NotNull ODouble radicand,
        @NotNull ODouble exponent) {
        return POWER.on(radicand, new ODouble(1/exponent.getDouble()));
    }

    @Contract(pure = true) @NotNull ODouble on(@NotNull ODouble radicand,
        @NotNull OFraction exponent){
        return POWER.on(radicand, new OFraction(1/exponent.getDouble()));
    }

    //region Fraction
    //
    // -->
    // -->
    //

    @Contract(pure = true) @NotNull OFraction on(@NotNull OFraction
        radicand, @NotNull ODouble exponent){
        return POWER.on(radicand, new OFraction(1/exponent.getDouble()));
    }

    @Contract(pure = true) @NotNull OFraction on(@NotNull OFraction
        radicand, @NotNull OFraction exponent){
        return POWER.on(radicand, new ODouble(1/exponent.getDouble()));
    }

    //region Matrix
    //
    // -->
    // -->
    //

    @Contract(pure = true) @NotNull OMatrix on(@NotNull OMatrix radicand,
        @NotNull ODouble exponent) {
        return POWER.on(radicand, new ODouble(1/exponent.getDouble()));
    }

    @Contract(pure = true) @NotNull OMatrix on(@NotNull OMatrix radicand,
        @NotNull OFraction exponent) {
        return POWER.on(radicand, new ODouble(1/exponent.getDouble()));
    }
}

```

Listing 35: RootTest (Pham)

```
package de.fhdw.wip.rpntilecalculator.model.calculation;
```

```

import org.junit.Test;
import static org.junit.Assert.*;

import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import de.fhdw.wip.rpntilecalculator.model.operands.OFraction;

/*
 * Summary: Test for the class 'RootTest'
 * Author: Khang Pham
 */
public class RootTest {

    private Root ROOT = Root.getInstance();

    //region Double
    @Test public void onDoubleDouble_isCorrect() {
        assertTrue(ROOT.on(
            new ODouble(36),
            new ODouble(2)
        ).equalsValue(new ODouble((6))));

    }

    @Test public void onDoubleFraction_isCorrect() {
        assertTrue(ROOT.on(
            new ODouble(36),
            new OFraction(0.5)
        ).equalsValue(new ODouble((1296))));

    }
}

```

10.1.2 Operands

Listing 36: DoubleComparator (Schwenke)

```

package de.fhdw.wip.rpntilecalculator.model.operands;

import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

import java.util.Arrays;
import java.util.Set;

/*
 * Summary: Utility to compare Doubles.
 * Author: Tim Schwenke
 */
public class DoubleComparator {

    /**
     * Check if two doubles are equal (with a certain margin).
     * @param d1 First double
     * @param d2 Second double
     * @return Boolean
     */
    @Contract(pure = true) public static boolean isEqual(double d1, double

```

```

    ↗ d2) {
    return Math.abs(d1 - d2) < 0.000001;
}

/**
 * Compares two arrays of Double.
 * @param d1 First double array.
 * @param d2 Second double array
 * @return Boolean
 */
@Contract(pure = true) public static boolean isEqual(
    @NotNull double[] d1,
    @NotNull double[] d2
) {
    if (d1.length != d2.length) return false;
    if (Arrays.equals(d1, d2)) return true;

    for (int i = 0; i < d1.length; i++)
        if (!isEqual(d1[i], d2[i])) return false;

    return true;
}

/**
 * Compares two matrices of doubles for equality.
 * @param d1 First matrix
 * @param d2 Second matrix
 * @return Boolean
 */
@Contract(pure = true) public static boolean isEqual(
    @NotNull double[][] d1,
    @NotNull double[][] d2
) {
    if (d1.length != d2.length) return false;
    for (int i = 0; i < d1.length; i++)
        if (d1[i].length != d2[i].length) return false;

    for (int i = 0; i < d1.length; i++)
        if (!isEqual(d1[i], d2[i])) return false;

    return true;
}

/**
 * Compares two sets of Doubles for equality.
 * @param d1 First set
 * @param d2 Second set
 * @return Boolean
 */
@Contract(pure = true) public static boolean isEqual(
    @NotNull Set<Double> d1,
    @NotNull Set<Double> d2
) {
    Double[] d1Array = new Double[d1.size()];
    d1.toArray(d1Array);
}

```

```
Double[] d2Array = new Double[d2.size()];
d2.toArray(d2Array);

double[] d1ArrayPrim = new double[d1.size()];
for (int i = 0; i < d1.size(); i++) d1ArrayPrim[i] = d1Array[i];

double[] d2ArrayPrim = new double[d2.size()];
for (int i = 0; i < d1.size(); i++) d2ArrayPrim[i] = d2Array[i];

return isEqual(d1ArrayPrim, d2ArrayPrim);
}

/**
 * Check if double is about zero (margin).
 * @param d Double to check
 * @return Boolean
 */
@Contract(pure = true) public static boolean isZero(double d) {
    double delta = 0.00001;
    return d < delta && d > -1d * delta;
}
}
```

Listing 37: DoubleFormatter (Schwenke)

```
package de.fhdw.wip.rpntilecalculator.model.operands;

import org.jetbrains.annotations.NotNull;

import java.text.DecimalFormat;

/*
 * Summary: Util for formatting all Double Values
 * Author: Tim Schwenke
 */
public final class DoubleFormatter {

    // DecimalFormat
    private static final DecimalFormat DF = new DecimalFormat("#.##");

    /*
     * Format the given double in the format to a string
     * @param d the double that is to be formatted
     * @return the formatted string
     */
    @NotNull public static String format(double d) {
        return DF.format(d);
    }
}
```

Listing 38: Element (Schwenke)

```
package de.fhdw.wip.rpntilecalculator.model.operands;
```

```

/*
 * Summary: Main Element Wrapper for all kinds
 * Author: Tim Schwenke
 */
public abstract class Element {
}

```

Listing 39: ODouble (Schwenke)

```

package de.fhdw.wip.rpntilecalculator.model.operands;

import org.apache.commons.math3.primes.Primes;

import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;

/*
 * Summary: Wrapper for the Double Operand
 * Author: Tim Schwenke
 */
public class ODouble extends Operand {

    // content to be wrapped
    private double aDouble;

    /*
     * Create a new ODouble from a double
     * @param aDouble content to be wrapped
     */
    public ODouble(double aDouble) {
        this.aDouble = aDouble;
    }

    /*
     * Create a new ODouble from a string
     * @param aDouble content to be wrapped
     */
    public ODouble(String aDouble) { this.aDouble = Double.valueOf(aDouble
        ↗ ); }

    /*
     * Get the underlying content
     * @return the underlying content
     */
    public double getDouble() {
        return aDouble;
    }

    /*
     * swap the pre Sign (+ -> -; - -> +)
     * @return new double
     */
    @NotNull @Override public ODouble turnAroundSign() {
        return new ODouble(aDouble * -1);
    }
}

```

```

    }

    /*
     * Turn the pre Sign to negative
     * @return new double
     */
    @NotNull @Override public ODouble negateValue() {
        return new ODouble(Math.abs(aDouble) * -1);
    }

    /*
     * Inverse the value
     * @return new double
     */
    @NotNull @Override public ODouble inverseValue() {
        return new ODouble(1 / aDouble);
    }

    /*
     * Format the double to a string
     * @return String representation of the content
     */
    @NotNull @Override public String toString() {
        return DoubleFormatter.format(aDouble);
    }

    @Override public boolean equalsValue(@Nullable Operand operand) {
        if (operand == this) return true;
        if (!(operand instanceof ODouble)) return false;

        double bDouble = ((ODouble) operand).getDouble();

        return DoubleComparator.isEqual(aDouble, bDouble);
    }

    /*
    If number has a decimal part it returns false
    */
    public boolean isPrime()
    {
        if(this.aDouble % 1 == 0)
        {
            return Primes.isPrime((int) this.aDouble);
        }
        else
        {
            return false;
        }
    }
}

```

Listing 40: OEmpty (Meinerzhagen)

```
package de.fhdw.wip.rpntilecalculator.model.operands;
```

```

import org.apache.commons.math3.primes.Primes;
import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;

/*
 * Summary: Wrapper for the Double Operand
 * Author: Tim Jonas Meinerzhagen
 */
public class OEmpty extends Operand {

    /*
     * Create a an empty operand
     * @param aDouble content to be wrapped
     */
    public OEmpty() { }

    /*
     * Create a an empty operand
     * @param aDouble content to be wrapped
     */
    public OEmpty(String content) { }

    /*
     * swap the pre Sign
     * @return new empty
     */
    @NotNull @Override public OEmpty turnAroundSign() { return this; }

    /*
     * Turn the pre Sign to negative
     * @return new empty
     */
    @NotNull @Override public OEmpty negateValue() { return this; }

    /*
     * Inverse the value
     * @return new empty
     */
    @NotNull @Override public OEmpty inverseValue() { return this; }

    /*
     * Format the empty to a string
     * @return String representation of the content
     */
    @NotNull @Override public String toString() { return " "; }

    @Override public boolean equalsValue(@Nullable Operand operand) {
        if (operand == this) return true;
        return operand instanceof OEmpty;
    }
}

```

Listing 41: OFraction (Schwenke)

```

package de.fhdw.wip.rpntilecalculator.model.operands;

import org.apache.commons.math3.fraction.Fraction;
import org.apache.commons.math3.primes.Primes;
import org.jetbrains.annotations.NotNull;

/*
 * Summary: Wrapper for the Fraction Operand
 * Author: Tim Schwenke
 */
public class OFraction extends Operand {

    @NotNull private Fraction fraction;

    public OFraction(@NotNull Fraction fraction) {
        this.fraction = fraction;
    }

    public OFraction(int nom, int den) {
        this.fraction = new Fraction(nom, den);
    }

    public OFraction(@NotNull double doubleValue) {this.fraction = new
        ↪ Fraction(doubleValue); }

    public OFraction(@NotNull String fraction) {
        String[] vars = fraction.split("[(/)]");
        int nom = Integer.valueOf(vars[1]);
        int den = Integer.valueOf(vars[2]);
        this.fraction = new Fraction(nom, den);
    }

    public @NotNull Fraction getFraction() {
        return fraction;
    }

    public double getDouble() {
        return fraction.doubleValue();
    }

    @NotNull @Override public OFraction turnAroundSign() {
        return new OFraction(fraction.multiply(-1));
    }

    @NotNull @Override public OFraction negateValue() {
        return new OFraction(new Fraction(
            Math.abs(fraction.getNumerator()) * -1,
            Math.abs(fraction.getDenominator()) * -1
        ));
    }

    @Override public @NotNull OFraction inverseValue() {
        return new OFraction(fraction.reciprocal());
    }
}

```

```

@Override
public boolean equalsValue(Operand operand) {
    if (operand == this) return true;
    if (!(operand instanceof OFraction)) return false;

    return ((OFraction) operand).getFraction().compareTo(fraction) ==
        ↪ 0;
}

@NotNull @Override public String toString() {
    return String.format("(%s/%s)",
        DoubleFormatter.format(fraction.getNumerator()),
        DoubleFormatter.format(fraction.getDenominator()))
    );
}

/*
If number has a decimal part it returns false
For the case if the Fraction is natural number
*/
public boolean isPrime()
{
    double doubleValue = getDouble();
    if(doubleValue % 1 == 0)
    {
        return Primes.isPrime((int) doubleValue);
    }
    else
    {
        return false;
    }
}

```

Listing 42: OMatrix (Schwenke)

```

package de.fhdw.wip.rpntilecalculator.model.operands;

import org.apache.commons.math3.linear.Array2DRowRealMatrix;
import org.apache.commons.math3.linear.MatrixUtils;
import org.apache.commons.math3.linear.RealMatrix;
import org.jetbrains.annotations.NotNull;

import java.util.ArrayList;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/*
 * Summary: Wrapper for the Matrix Operand
 * Author: Tim Schwenke
 */
@SuppressWarnings("unused")
public class OMatrix extends Operand {

```

```

@NotNull private RealMatrix matrix;

public OMatrix(@NotNull RealMatrix matrix) {
    this.matrix = matrix;
}

public OMatrix(@NotNull double[][] doubleMatrix) {
    int longest = 0;

    for (double[] dim1 : doubleMatrix)
        if (dim1.length > longest) longest = dim1.length;

    double[][] modified = new double[doubleMatrix.length][longest];
    for (int i = 0; i < doubleMatrix.length; i++)
        System.arraycopy(
            doubleMatrix[i], 0,
            modified[i], 0, doubleMatrix[i].length
        );

    matrix = new Array2DRowRealMatrix(modified);
}

public OMatrix(@NotNull String matrix) {
    //[[1.23, 1.32], [0.23, 1.23]]
    ArrayList<double[]> listMatrix = new ArrayList<>();
    Pattern pat1 = Pattern.compile("\\[[^\\[\\]]+\\]\\.\\*?\\]");
    Matcher mat1 = pat1.matcher(matrix);

    while(mat1.find()) {
        String row = matrix.substring(mat1.start(), mat1.end());

        ArrayList<Double> listArray = new ArrayList<>();
        pat1 = Pattern.compile("[\\-0-9.]+");
        Matcher mat2 = pat1.matcher(row);

        while(mat2.find()) {
            String value = row.substring(mat2.start(), mat2.end());
            listArray.add(Double.valueOf(value));
        }

        double[] doubleArray = new double[listArray.size()];
        for(int i = 0; i < listArray.size(); i++) doubleArray[i] =
            listArray.get(i);
        listMatrix.add(doubleArray);
    }
    double[][] doubleMatrix = new double[listMatrix.size()][];
    for(int i = 0; i < listMatrix.size(); i++) doubleMatrix[i] =
        listMatrix.get(i);
    this.matrix = new Array2DRowRealMatrix(doubleMatrix);
}

public @NotNull RealMatrix getMatrix() {
    return matrix;
}

```

```

    @NotNull @Override public OMatrix turnAroundSign() {
        return new OMatrix(matrix.scalarMultiply(-1));
    }

    @NotNull @Override public OMatrix negateValue() {
        double[][] dim1 = matrix.getData();
        for (int i = 0; i < dim1.length; i++)
            for (int k = 0; k < dim1[i].length; k++)
                dim1[i][k] = Math.abs(dim1[i][k]) * -1;
        return new OMatrix(new Array2DRowRealMatrix(dim1));
    }

    @Override public @NotNull OMatrix inverseValue() {
        return new OMatrix(MatrixUtils.inverse(matrix));
    }

    @Override
    public boolean equalsValue(Operand operand) {
        if (operand == this) return true;
        if (!(operand instanceof OMatrix)) return false;

        return DoubleComparator.isEqual(
            matrix.getData(),
            ((OMatrix) operand).getMatrix().getData()
        );
    }

    @NotNull @Override public String toString() {
        StringBuilder builder = new StringBuilder();
        builder.append("[");
        for (double[] doubles : matrix.getData()) {
            builder.append("[");
            for (double d : doubles) {
                builder.append(DoubleFormatter.format(d));
                builder.append(", ");
            }
            builder.delete(builder.length() - 2, builder.length());
            builder.append("], ");
        }
        builder.delete(builder.length() - 2, builder.length());
        builder.append("]");
        return builder.toString();
    }
}

```

Listing 43: OPolynom (Schwenke)

```

package de.fhdw.wip.rpntilecalculator.model.operands;

import org.apache.commons.math3.analysis.polynomials.PolynomialFunction;
import org.jetbrains.annotations.NotNull;

import java.util.ArrayList;

```

```

/*
 * Summary: Wrapper for the Polynom Operand
 * Author: Tim Schwenke
 */
public class OPolynom extends Operand {

    @NotNull private PolynomialFunction polynom;

    public OPolynom(@NotNull PolynomialFunction polynom) {
        this.polynom = polynom;
    }

    public OPolynom(@NotNull double... coefficients) {
        this.polynom = new PolynomialFunction(coefficients);
    }

    public OPolynom(@NotNull String polynom) {
        //4.1x^0 + 2x^1 + -3.1x^2
        String[] vars = polynom.trim().split("(x\\^\\d+)( \\+)*");
        double[] coefficients = new double[vars.length];
        for(int i = 0; i < vars.length; i++) coefficients[i] = Double.parseDouble(vars[i].trim());
        this.polynom = new PolynomialFunction(coefficients);
    }

    public @NotNull PolynomialFunction getPolynom() {
        return polynom;
    }

    @NotNull @Override public OPolynom turnAroundSign() {
        double[] doubles = polynom.getCoefficients();
        for (int i = 0; i < doubles.length; i++)
            doubles[i] *= -1;
        return new OPolynom(new PolynomialFunction(doubles));
    }

    @NotNull @Override public OPolynom negateValue() {
        double[] doubles = polynom.getCoefficients();
        for (int i = 0; i < doubles.length; i++)
            doubles[i] = Math.abs(doubles[i]) * -1;
        return new OPolynom(new PolynomialFunction(doubles));
    }

    @Override public @NotNull OPolynom inverseValue() {
        double[] doubles = polynom.getCoefficients();
        for (int i = 0; i < doubles.length; i++)
            doubles[i] = 1 / doubles[i];
        return new OPolynom(new PolynomialFunction(doubles));
    }

    @Override
    public boolean equalsValue(Operand operand) {
        if (operand == this) return true;
        if (!(operand instanceof OPolynom)) return false;
    }
}

```

```

        return DoubleComparator.isEqual(
            polynom.getCoefficients(),
            ((OPolynom) operand).getPolynom().getCoefficients()
        );
    }

    @NotNull @Override public String toString() {
        StringBuilder builder = new StringBuilder();
        double[] doubles = polynom.getCoefficients();
        for (int i = 0; i < doubles.length; i++) {
            builder.append(DoubleFormatter.format(doubles[i]));
            builder.append("x^");
            builder.append(i);
            builder.append(" + ");
        }
        builder.delete(builder.length() - 3, builder.length());
        return builder.toString();
    }

    @NotNull public OPolynom getDerivative()
    {
        PolynomialFunction polynomialDerivative = polynom.
            ↪ polynomialDerivative();
        return new OPolynom(polynomialDerivative);
    }
}

```

Listing 44: OSet (Schwenke)

```

package de.fhdw.wip.rpntilecalculator.model.operands;

import org.jetbrains.annotations.NotNull;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.Set;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/*
 * Summary: Every entry can only exist one time
 * Author: Tim Schwenke
 */
public class OSet extends Operand {

    @NotNull private Set<Double> set;

    /**
     * Create Set from set.
     * @param set Set
     */
    public OSet(@NotNull Set<Double> set) {
        this.set = set;
    }
}

```

```

    }

    /**
     * Create OSet from array of doubles
     * @param doubles Double array
     */
    public OSet(@NotNull double... doubles) {
        ArrayList<Double> list = new ArrayList<>();
        for (double d : doubles) list.add(d);

        this.set = new HashSet<>();
        this.set.addAll(list);
    }

    /**
     * Create OSet from String
     * @param set String representation of the Set
     */
    public OSet(@NotNull String set) {
        this.set = new HashSet<>();
        Pattern pat = Pattern.compile("[\\-0-9.]+");
        Matcher mat = pat.matcher(set);

        while(mat.find()) {
            this.set.add(Double.valueOf(set.substring(mat.start(), mat.end
                ↪ ())));
        }
    }

    /**
     * Get underlying Set
     * @return Set
     */
    @NotNull public Set<Double> getSet() {
        return set;
    }

    /**
     * Turn around all signs
     * @return New OSet
     */
    @NotNull @Override public OSet turnAroundSign() {
        Set<Double> newSet = new HashSet<>();
        for (double d : set)
            newSet.add(d * -1);
        return new OSet(newSet);
    }

    /**
     * Negate all values
     * @return New OSet
     */
    @NotNull @Override public OSet negateValue() {
        Set<Double> newSet = new HashSet<>();
        for (double d : set)

```

```

        newSet.add(Math.abs(d) * -1);
    return new OSet(newSet);
}

/**
 * Inverse all values
 * @return new OSet
 */
@Override public @NotNull OSet inverseValue() {
    Set<Double> newSet = new HashSet<>();
    for (double d : set)
        newSet.add(1 / d);
    return new OSet(newSet);
}

/**
 * Compare this instance with another Operand
 * @param operand Another operand
 * @return Boolean
 */
@Override public boolean equalsValue(Operand operand) {
    if (operand == this) return true;
    if (!(operand instanceof OSet)) return false;

    return DoubleComparator.isEqual(set, ((OSet) operand).getSet());
}

/**
 * Turn this instance into an string.
 * @return String
 */
@NotNull @Override public String toString() {
    StringBuilder builder = new StringBuilder();
    builder.append("[");
    for (double d : set) {
        builder.append(DoubleFormatter.format(d));
        builder.append(", ");
    }
    builder.delete(builder.length() - 2, builder.length());
    builder.append("]");
    return builder.toString();
}
}

```

Listing 45: OTuple (Schwenke)

```

package de.fhdw.wip.rpntilecalculator.model.operands;

import org.jetbrains.annotations.NotNull;
import java.util.ArrayList;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

```

```

/*
 * Summary: Wrapper for the Tuple Operand
 * Author: Tim Schwenke
 */
public class OTuple extends Operand {

    @NotNull private double[] tuple;

    /**
     * Create tuple from array of doubles.
     * @param doubles
     */
    public OTuple(@NotNull double... doubles) {
        this.tuple = doubles;
    }

    /**
     * Create tuple from list of doubles
     * @param tuple
     */
    private OTuple(@NotNull List<Double> tuple) {
        this.tuple = new double[tuple.size()];
        for (int i = 0; i < this.tuple.length; i++)
            this.tuple[i] = tuple.get(i);
    }

    /**
     * Create Tuple from String
     * @param tuple Tuple as String
     */
    public OTuple(@NotNull String tuple) {
        ArrayList<Double> listTuple = new ArrayList<>();
        Pattern pat = Pattern.compile("[\\-0-9.]+");
        Matcher mat = pat.matcher(tuple);

        while(mat.find()) {
            String value = tuple.substring(mat.start(), mat.end());
            listTuple.add(Double.valueOf(value));
        }
        this.tuple = new double[2];
        for(int i = 0; i < 2; i++) this.tuple[i] = listTuple.get(i);
    }

    /**
     * Get the underlying Tuple.
     * @return Tuple
     */
    public @NotNull double[] getTuple() {
        return tuple;
    }

    /**
     * Turn around all signs.
     * @return new Tuple.
     */
}

```

```

@NotNull @Override public OTuple turnAroundSign() {
    List<Double> newTuple = new ArrayList<>();
    for (double d : tuple)
        newTuple.add(d * -1);
    return new OTuple(newTuple);
}

/**
 * Negate Value. Make all values negative.
 * @return new Tuple
 */
@NotNull @Override public OTuple negateValue() {
    List<Double> newTuple = new ArrayList<>();
    for (double d : tuple)
        newTuple.add(Math.abs(d) * -1);
    return new OTuple(newTuple);
}

/**
 * Inverse the value of this instance
 * @return New Tuple
 */
@Override
public @NotNull OTuple inverseValue() {
    List<Double> newTuple = new ArrayList<>();
    for (double d : tuple)
        newTuple.add(1 / d);
    return new OTuple(newTuple);
}

/**
 * Compare this instance with another Operand
 * @param operand Another operand.
 * @return Boolean
 */
@Override
public boolean equalsValue(Operand operand) {
    if (operand == this) return true;
    if (!(operand instanceof OTuple)) return false;

    return DoubleComparator.isEqual(tuple, ((OTuple) operand).getTuple
        ↲ ());
}

/**
 * Turn Operand into String representation
 * @return String
 */
@NotNull @Override public String toString() {
    StringBuilder builder = new StringBuilder();
    builder.append("(");
    for (double d : tuple) {
        builder.append(DoubleFormatter.format(d));
        builder.append(", ");
    }
}

```

```
        builder.delete(builder.length() - 2, builder.length());
        builder.append(")");
        return builder.toString();
    }

}
```

Listing 46: Operand (Schwenke)

```
package de.fhdw.wip.rpntilecalculator.model.operands;

import org.jetbrains.annotations.NotNull;

/*
 * Summary: Main class for all operands that can be used for calculating
 * Author: Tim Schwenke
 */
public abstract class Operand extends Element {

    /**
     * Multiplies all values of the {@link Operand} with {@code -1}.
     */
    public abstract @NotNull Operand turnAroundSign();

    /**
     * Makes all values of the {@link Operand} negative.
     */
    public abstract @NotNull Operand negateValue();

    public abstract @NotNull Operand inverseValue();

    public abstract boolean equalsValue(Operand operand);

}
```

10.1.3 Settings

Listing 47: AllClear (Falk)

```
package de.fhdw.wip.rpntilecalculator.model.settings;

import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

import de.fhdw.wip.rpntilecalculator.presenter.Presenter;

/**
 * Summary: Empties the stack of the presenter
 * Author: Hendrik Falk
 */
public class AllClear extends Setting {

    @Contract(pure = true) @NotNull
    public static AllClear getInstance() { return new AllClear(); }
```

```
    /**
     * Clears the entire Stack and current input
     */
    @Override
    public boolean call() {
        Presenter.OPERAND_STACK.clear();
        Presenter.resetInputTerm(null);
        Presenter.updateStack();
        return true;
    }
}
```

Listing 48: ClearHistory (Falk)

```
package de.fhdw.wip.rpntilecalculator.model.settings;

import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

import de.fhdw.wip.rpntilecalculator.presenter.Presenter;

/**
 * Summary: Empties the history stack of the presenter
 * Author: Hendrik Falk
 */
public class ClearHistory extends Setting {

    @Contract(pure = true) @NotNull
    public static ClearHistory getInstance() { return new ClearHistory();
    }

    /**
     * Clears the entire History
     */
    @Override
    public boolean call() {
        Presenter.HISTORY_STACK.clear();
        Presenter.updateHistoryStack();
        return true;
    }
}
```

Listing 49: DeleteEntry (Falk)

```
package de.fhdw.wip.rpntilecalculator.model.settings;

import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

import de.fhdw.wip.rpntilecalculator.presenter.Presenter;

/**
 * Summary: Deletes the last input of the stack
 * Author: Hendrik Falk
 */
```

```

public class DeleteEntry extends Setting {

    @Contract(pure = true) @NotNull
    public static DeleteEntry getInstance() { return new DeleteEntry(); }

    /**
     * Delete the last entry and resetting the input term
     */
    @Override
    public boolean call() {
        Presenter.OPERAND_STACK.pop();
        Presenter.resetInputTerm(Presenter.OPERAND_STACK.peek());
        Presenter.updateStack();
        return true;
    }
}

```

Listing 50: Dot (Falk)

```

package de.fhdw.wip.rpntilecalculator.model.settings;

import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import de.fhdw.wip.rpntilecalculator.presenter.Presenter;

/**
 * Summary: Places a '.' in the input term to create decimal values
 * Author: Hendrik Falk
 */
public class Dot extends Setting {

    @Contract(pure = true) @NotNull
    public static Dot getInstance() { return new Dot(); }

    @Override
    public boolean call() {
        if(Presenter.INPUT_TERM.toString().equals(Presenter.
            ↪ INPUT_FINALIZED)) {
            ODouble oDouble = new ODouble(0);
            Presenter.resetInputTerm(oDouble);
            Presenter.OPERAND_STACK.push(oDouble);
        }
        if(!Presenter.INPUT_TERM.toString().contains(".")) Presenter.
            ↪ INPUT_TERM.append(".");
        Presenter.updateStack();
        return true;
    }
}

```

Listing 51: Enter (Falk)

```
package de.fhdw.wip.rpntilecalculator.model.settings;
```

```

import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

import de.fhdw.wip.rpntilecalculator.model.operands.Operand;
import de.fhdw.wip.rpntilecalculator.presenter.Presenter;

/**
 * Summary: Finishes the current input term so that a new input can be
 *          ↪ created
 * Author: Hendrik Falk
 */
public class Enter extends Setting {

    @Contract(pure = true) @NotNull
    public static Enter getInstance() { return new Enter(); }

    /**
     * finalizes an input string
     */
    @Override
    public boolean call() {
        Presenter.INPUT_TERM = new StringBuilder().append(Presenter.
            ↪ INPUT_FINALIZED);
        if(Presenter.OPERAND_STACK.size() != 0) {
            Presenter.add2History(Presenter.OPERAND_STACK.peek());
            Presenter.updateHistoryStack();
        }
        return true;
    }
}

```

Listing 52: Inverse (Falk)

```

package de.fhdw.wip.rpntilecalculator.model.settings;

import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

import de.fhdw.wip.rpntilecalculator.presenter.Presenter;
import de.fhdw.wip.rpntilecalculator.model.operands.Operand;

/**
 * Summary: Calculates the inverse of an operand
 * Author: Hendrik Falk
 */
public class Inverse extends Setting {

    @Contract(pure = true) @NotNull
    public static Inverse getInstance() { return new Inverse(); }

    /**
     * Changes + values to - and vice versa
     */
    @Override
    public boolean call() {
        if(Presenter.OPERAND_STACK.size() == 0) return false;
    }
}

```

```

        Operand operand = Presenter.OPERAND_STACK.peek();
        Presenter.OPERAND_STACK.pop();
        Operand result = operand.inverseValue();
        Presenter.OPERAND_STACK.push(result);
        Presenter.resetInputTerm(result);
        Presenter.updateStack();
        return true;
    }
}

```

Listing 53: LoadLayout (Meinerzhagen)

```

package de.fhdw.wip.rpntilecalculator.model.settings;

import android.app.Dialog;
import android.content.DialogInterface;
import android.view.View;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.ListAdapter;
import android.widget.ListView;

import androidx.appcompat.app.AlertDialog;

import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

import de.fhdw.wip.rpntilecalculator.MainActivity;
import de.fhdw.wip.rpntilecalculator.view.layout.TileLayoutFactory;
import de.fhdw.wip.rpntilecalculator.view.layout.TileLayoutLoader;

/**
 * Summary: Creates a load layout menu to load a new layout design
 * Author: Tim Jonas Meinerzhagen
 */
public class LoadLayout extends Setting {

    @Contract(pure = true) @NotNull
    public static LoadLayout getInstance() { return new LoadLayout(); }

    /**
     * Clears the entire Stack and current input
     */
    @Override
    public boolean call() {
        final MainActivity activity = MainActivity.mainActivity;

        final Dialog dialog = new Dialog(activity);

        LinearLayout l = new LinearLayout(activity.getBaseContext());
        for(final String s : TileLayoutLoader.getSavedLayouts(activity.
            ↪ getBaseContext())){
            Button b = new Button(activity.getBaseContext());
            b.setText(s);
            b.setOnClickListener(new View.OnClickListener() {
                @Override

```

```

        public void onClick(View view) {
            dialog.cancel();
            activity.setTileLayout(TileLayoutFactory.createLayout(
                ↪ activity.getBaseContext(), s));
        }
    });
    l.addView(b);
}
dialog.addContentView(l,
    new LinearLayout.LayoutParams(LinearLayout.LayoutParams.
        ↪ WRAP_CONTENT, LinearLayout.LayoutParams.WRAP_CONTENT)
    ↪ );
dialog.setCancelable(true);
dialog.show();

return true;
}

```

Listing 54: SaveLayout (Meinerzhagen)

```

package de.fhdw.wip.rpntilecalculator.model.settings;

import android.app.Dialog;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.LinearLayout;

import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

import de.fhdw.wip.rpntilecalculator.MainActivity;
import de.fhdw.wip.rpntilecalculator.view.Tile;
import de.fhdw.wip.rpntilecalculator.view.layout.TileLayout;
import de.fhdw.wip.rpntilecalculator.view.layout.TileLayoutFactory;
import de.fhdw.wip.rpntilecalculator.view.layout.TileLayoutLoader;

/**
 * Summary: Creates a save layout menu to save the current design
 * Author: Tim Jonas Meinerzhagen
 */
public class SaveLayout extends Setting {

    @Contract(pure = true) @NotNull
    public static SaveLayout getInstance() { return new SaveLayout(); }

    /**
     * Clears the entire Stack and current input
     */
    @Override
    public boolean call() {
        final MainActivity activity = MainActivity.mainActivity;
        final Dialog dialog = new Dialog(activity);

```

```
    LinearLayout l = new LinearLayout(activity.getBaseContext());
    final EditText text = new EditText(activity.getBaseContext());
    text.setText("Main");
    Button b = new Button(activity.getBaseContext());
    b.setText("Save");
    b.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            dialog.cancel();
            TileLayout t = activity.getTileLayout();
            t.setIndicator(text.getText().toString());
            TileLayoutLoader.saveLayout(activity.getBaseContext(), t);
            t.showAnimation(Tile.buttonSave);
        }
    });
    l.addView(text);
    l.addView(b);
    dialog.addContentView(l,
        new LinearLayout.LayoutParams(LinearLayout.LayoutParams.WRAP_CONTENT,
            LinearLayout.LayoutParams.WRAP_CONTENT));
    dialog.setCancelable(true);
    dialog.show();

    return true;
}
}
```

Listing 55: Setting (Falk)

```
package de.fhdw.wip.rpntilecalculator.model.settings;

import org.jetbrains.annotations.Contract;

/**
 * Summary: Super class for settings
 * Author: Hendrik Falk
 */
public abstract class Setting {

    @Contract(pure = true)
    public abstract boolean call();
}
```

Listing 56: Split (Falk)

```
package de.fhdw.wip.rpntilecalculator.model.settings;

import org.apache.commons.math3.linear.RealMatrix;
import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

import java.util.ArrayList;
import java.util.Collections;
```

```

import java.util.List;
import java.util.Set;

import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import de.fhdw.wip.rpntilecalculator.model.operands.OMatrix;
import de.fhdw.wip.rpntilecalculator.model.operands.OTuple;
import de.fhdw.wip.rpntilecalculator.model.operands.Operand;
import de.fhdw.wip.rpntilecalculator.presenter.Presenter;

/**
 * Summary: Splits the current loaded operand into several ODouble values
 * Author: Hendrik Falk
 */
public class Split extends Setting {

    @Contract(pure = true) @NotNull
    public static Split getInstance() { return new Split(); }

    /**
     * Splits a set / matrix / vektor / tuple into multiple ODouble values
     */
    @Override
    public boolean call() {
        if(Presenter.OPERAND_STACK.size() == 0) return false;

        Operand toSplit = Presenter.OPERAND_STACK.peek();
        List<ODouble> operandList = new ArrayList<>();

        if(toSplit instanceof OMatrix) {

            RealMatrix matrix = ((OMatrix) toSplit).getMatrix();
            int rowsCount = matrix.getRowDimension();
            for(int i = 0; i < rowsCount; i++) {
                double[] row = matrix.getRow(i);
                for(double value : row) operandList.add(new ODouble(value))
                    ↘ ;
            }
        } else if(toSplit instanceof OSet) {

            @NotNull Set<Double> set = ((OSet) toSplit).getSet();
            for(double value : set) operandList.add(new ODouble(value));
        } else if(toSplit instanceof OTuple) {

            double[] tuple = ((OTuple) toSplit).getTuple();
            for(double value : tuple) operandList.add(new ODouble(value));
        }

        if(operandList.size() != 0) {
            Presenter.OPERAND_STACK.pop();

            Collections.reverse((operandList));
        }
    }
}

```

```

        for(ODouble oDouble : operandList) {
            Presenter.add2History(oDouble);
            Presenter.OPERAND_STACK.push(oDouble);
        }

        Presenter.updateStack();
        Presenter.updateHistoryStack();
    }
    return true;
}
}

```

Listing 57: Swap (Falk)

```

package de.fhdw.wip.rpntilecalculator.model.settings;

import org.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

import de.fhdw.wip.rpntilecalculator.presenter.Presenter;
import de.fhdw.wip.rpntilecalculator.model.operands.Operand;

/**
 * Summary: Swaps the last two stack operands
 * Author: Hendrik Falk
 */
public class Swap extends Setting {

    @Contract(pure = true) @NotNull
    public static Swap getInstance() { return new Swap(); }

    /**
     * Swap the last entry with the one before
     */
    @Override
    public boolean call() {
        if(Presenter.OPERAND_STACK.size() < 2) return false;

        Operand one = Presenter.OPERAND_STACK.pop();
        Operand two = Presenter.OPERAND_STACK.pop();

        Presenter.OPERAND_STACK.push(one);
        Presenter.OPERAND_STACK.push(two);
        Presenter.resetInputTerm(two);

        Presenter.updateStack();
        return true;
    }
}

```

Listing 58: TurnAroundSign (Falk)

```

package de.fhdw.wip.rpntilecalculator.model.settings;

import org.jetbrains.annotations.Contract;

```

```

import org.jetbrains.annotations.NotNull;

import de.fhdw.wip.rpntilecalculator.presenter.Presenter;
import de.fhdw.wip.rpntilecalculator.model.operands.Operand;

/**
 * Summary: Changes positive operands to negative ones and vice versa
 * Author: Hendrik Falk
 */
public class TurnAroundSign extends Setting {

    @Contract(pure = true) @NotNull
    public static TurnAroundSign getInstance() { return new TurnAroundSign
        () ; }

    /**
     * Changes + values to - and vice versa
     */
    @Override
    public boolean call() {
        if(Presenter.OPERAND_STACK.size() == 0) return false;
        Operand operand = Presenter.OPERAND_STACK.peek();
        Presenter.OPERAND_STACK.pop();
        Operand result = operand.turnAroundSign();
        Presenter.OPERAND_STACK.push(result);
        Presenter.resetInputTerm(result);
        Presenter.updateStack();
        return true;
    }
}

```

10.1.4 Stack

Listing 59: StackInterface (Keienburg)

```

package de.fhdw.wip.rpntilecalculator.model.stack;

import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;

import java.util.List;

/**
 * Summary: Stack interface for stacks in this project
 * Author: Jannis Luca Keienburg
 * Date: 2020/01/03 & 2020/01/04
 */
public interface StackInterface<T> {

    /**
     * Push a single object onto the stack
     * @param value Object
     */
    void push(@NotNull T value);
}

```

```

/**
 * Push an array of objects onto the stack
 * @param values Array of objects
 */
void push(@NotNull T[] values);

/**
 * Pop an object from the stack
 * @return Object
 */
@Nullable T pop();

/**
 * Pop a certain amount of objects from the stack
 * @param max Max amount of objects to pop
 * @return List of popped items
 */
@NotNull List<T> pop(int max);

/**
 * Pop a element of a certain class from the stack
 * @param type Type of the element to be popped
 * @param <G> Class the type to be popped should extend
 * @return Popped item
 */
@Nullable <G extends T> G pop(Class<G> type);

/**
 * Pop a certain amount of elements of a certain type from the stack.
 * @param max Max amount of items to pop
 * @param type Type the items should be
 * @param <G> Class the items should extend
 * @return List of popped items
 */
@NotNull <G extends T> List<G> pop(int max, Class<G> type);

/**
 * Peek the first item on the stack
 * @return Item peeked
 */
@Nullable T peek();

/**
 * Peek a list of items on the stack.
 * @param max Max amount of items to peek
 * @return A list of items that were peeked
 */
@NotNull List<T> peek(int max);

/**
 * Peek the first item of a certain type on the stack.
 * @param type Type the item should be
 * @param <G> Class the item should extend
 * @return Peeked item
 */
@Nullable <G extends T> G peek(Class<G> type);

```

```

    /**
     * Peek a list of items that are all of a certain type on the stack
     * @param max Max amount of items to be peeked from the stack
     * @param type Type the peeked items should be
     * @param <G> Class the peeked items should extend
     * @return List of items that were peeked
     */
    @NotNull <G extends T> List<G> peek(int max, Class<G> type);

    /**
     * Check if a certain object is part of the stack
     * @param object Object to be searched for on the stack
     * @return Boolean
     */
    boolean contains(T object);

    /**
     * Clear the stack
     */
    void clear();

    /**
     * Get the stack as an copy in an array.
     * @return Array
     */
    @NotNull T[] get();

    /**
     * Get all items of a certain type from the stack
     * @param type Type the items should be
     * @param <G> Class items should extend
     * @return List
     */
    @NotNull <G extends T> List<G> get(Class<G> type);

    /**
     * Size of the stack
     * @return number of items on the stack
     */
    int size();
}

}

```

Listing 60: OperandStack (Keienburg)

```

package de.fhdw.wip.rpntilecalculator.model.stack;

import de.fhdw.wip.rpntilecalculator.model.operands.Operand;

import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;

import java.util.ArrayList;
import java.util.Deque;

```

```

import java.util.LinkedList;
import java.util.List;

/*
 * Summary: The Stack for the operands that is used for calculating
 * Author: Jannis Luca Keienburg
 * Date: 2020/01/03 & 2020/01/04
 */
public final class OperandStack implements StackInterface<Operand> {

    private final LinkedList<Operand> linkedList = new LinkedList<>();
    private final List<Operand> listView = linkedList;
    private final Deque<Operand> dequeView = linkedList;

    @Override public void push(@NotNull Operand operand) {
        dequeView.push(operand);
    }

    @Override public void push(@NotNull Operand[] operands) {
        for (Operand value : operands) dequeView.push(value);
    }

    @Override public @Nullable Operand pop() {
        try { return dequeView.pop(); }
        catch (RuntimeException e) { return null; }
    }

    @Override public @NotNull List<Operand> pop(int max) {
        List<Operand> list = new ArrayList<>();
        while (!linkedList.isEmpty() && list.size() < max)
            list.add(dequeView.pop());
        return list;
    }

    @Override public @Nullable <G extends Operand> G pop(Class<G> type) {
        for (int i = 0; i < listView.size(); i++) {
            if (type.isInstance(listView.get(i)))
                return type.cast(linkedList.remove(i));
        }
        return null;
    }

    @Override public @NotNull <G extends Operand> List<G> pop(int max,
        ↗ Class<G> type) {
        List<G> list = new ArrayList<>();
        for (int i = 0; i < linkedList.size(); i++) {
            if (list.size() < max && type.isInstance(linkedList.get(i))) {
                list.add(type.cast(linkedList.remove(i--)));
            }
        }
        return list;
    }

    @Override public @Nullable Operand peek() {
        try { return dequeView.peek(); }

```

```

        catch (RuntimeException e) { return null; }

    }

@Override public @NotNull List<Operand> peek(int max) {
    List<Operand> list = new ArrayList<>();
    for (int i = 0; i < listView.size(); i++)
        if (list.size() < max)
            list.add(listView.get(i));
    return list;
}

@Override public @Nullable <G extends Operand> G peek(Class<G> type) {
    for (int i = 0; i < listView.size(); i++) {
        Operand operand = listView.get(i);
        if (type.isInstance(operand))
            return type.cast(operand);
    }
    return null;
}

@Override public @NotNull <G extends Operand> List<G> peek(int max,
    ↪ Class<G> type) {
    List<G> list = new ArrayList<>();
    for (int i = 0; i < listView.size(); i++) {
        Operand operand = listView.get(i);
        if (list.size() < max && type.isInstance(operand))
            list.add(type.cast(operand));
    }
    return list;
}

@Override public boolean contains(Operand object) {
    return linkedList.contains(object);
}

@Override public void clear() {
    linkedList.clear();
}

@NotNull @Override public Operand[] get() {
    return linkedList.toArray(new Operand[0]);
}

@NotNull @Override public <G extends Operand> List<G> get(Class<G>
    ↪ type) {
    List<G> list = new ArrayList<>();
    for (Operand operand : listView)
        if (type.isInstance(operand))
            list.add(type.cast(operand));
    return list;
}

@Override public int size() {
    return linkedList.size();
}

```

```

    }

    public void print() {
        for (int i = 0; i < listView.size(); i++)
            System.out.println(i + ": " + listView.get(i));
    }

}

```

Listing 61: ArcCosinusTest (Keienburg)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import org.junit.Test;

import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;

import static org.junit.Assert.assertTrue;

/*
 * Summary: Unit test for the arc cosinus function
 * Author: Jannis Luca Keienburg
 * Date: 2020/01/22
 */

public class ArcCosinusTest {
    private ArcCosinus ARC_COSINUS = ArcCosinus.getInstance();

    @Test
    public void arcCosinusAngle_isCorrect1() {
        System.out.println(ARC_COSINUS.on(
            new ODouble(10)));
        assertTrue(ARC_COSINUS.on(
            new ODouble(10)).equalsValue(new ODouble
                ↪ (1.3953649341158527)))
    }

    @Test
    public void arcCosinusAngle_isCorrect2() {
        System.out.println(ARC_COSINUS.on(
            new ODouble(45)));
        assertTrue(ARC_COSINUS.on(
            new ODouble(45)).equalsValue(new ODouble
                ↪ (0.6674572160283838)))
    }

    @Test
    public void arcCosinusAngle_isCorrect3() {
        System.out.println(ARC_COSINUS.on(
            new ODouble(-45)));
        assertTrue(ARC_COSINUS.on(
            new ODouble(-45)).equalsValue(new ODouble
                ↪ (-0.6674572160283838)))
    }
}

```

```
    );
}
```

Listing 62: ArcSinusTest (Keienburg)

```
package de.fhdw.wip.rpntilecalculator.model.calculation;

import org.junit.Test;

import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;

import static org.junit.Assert.assertTrue;

/*
 * Summary: Unit test for the arc sinus function
 * Author: Jannis Luca Keienburg
 * Date: 2020/01/22
 */

public class ArcSinusTest {
    private ArcSinus ARC_SINUS = ArcSinus.getInstance();

    @Test
    public void arcSinusAngle_isCorrect1() {
        System.out.println(ARC_SINUS.on(
            new ODouble(10)));
        assertTrue(ARC_SINUS.on(
            new ODouble(10)).equalsValue(new ODouble
                ↪ (0.17543139267904395))
    );
}

    @Test
    public void arcSinusAngle_isCorrect2() {
        System.out.println(ARC_SINUS.on(
            new ODouble(45)));
        assertTrue(ARC_SINUS.on(
            new ODouble(45)).equalsValue(new ODouble
                ↪ (0.9033391107665127))
    );
}

    @Test
    public void arcSinusAngle_isCorrect3() {
        System.out.println(ARC_SINUS.on(
            new ODouble(-45)));
        assertTrue(ARC_SINUS.on(
            new ODouble(-45)).equalsValue(new ODouble
                ↪ (-0.9033391107665127))
    );
}
```

Listing 63: ArcTangensTest (Keienburg)

```
package de.fhdw.wip.rpntilecalculator.model.calculation;  
  
import org.junit.Test;  
  
import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;  
  
import static org.junit.Assert.assertTrue;  
  
/*  
 * Summary: Unit test for the arc tangens function  
 * Author: Jannis Luca Keienburg  
 * Date: 2020/01/22  
 */  
  
public class ArcTangensTest {  
    private ArcTangens ARC_TANGENS = ArcTangens.getInstance();  
  
    @Test  
    public void arcTangensAngle_isCorrect1() {  
        System.out.println(ARC_TANGENS.on(  
            new ODouble(10)));  
        assertTrue(ARC_TANGENS.on(  
            new ODouble(10)).equalsValue(new ODouble  
                ↪ (0.1727924348551592))  
    );  
    }  
  
    @Test  
    public void arcTangensAngle_isCorrect2() {  
        System.out.println(ARC_TANGENS.on(  
            new ODouble(45)));  
        assertTrue(ARC_TANGENS.on(  
            new ODouble(45)).equalsValue(new ODouble  
                ↪ (0.6657737500283538))  
    );  
    }  
  
    @Test  
    public void arcTangensAngle_isCorrect3() {  
        System.out.println(ARC_TANGENS.on(  
            new ODouble(-45)));  
        assertTrue(ARC_TANGENS.on(  
            new ODouble(-45)).equalsValue(new ODouble  
                ↪ (-0.6657737500283538))  
    );  
}
```

Listing 64: CosinusTest (Keienburg)

```
package de.fhdw.wip.rpntilecalculator.model.calculation;  
  
import org.junit.Test;
```

```

import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import static org.junit.Assert.assertTrue;

/*
 * Summary: Unit test for the cosinus function
 * Author: Jannis Luca Keienburg
 */

public class CosinusTest {
    private Cosinus COSINUS = Cosinus.getInstance();

    @Test public void cosinusAngle_isCorrect1() {
        System.out.println(COSINUS.on(
            new ODouble(10)));
        assertTrue(COSINUS.on(
            new ODouble(10)).equalsValue(new ODouble(0.984807753012208)
            ↪ )
    );
}

@Test public void cosinusAngle_isCorrect2() {
    System.out.println(COSINUS.on(
        new ODouble(45)));
    assertTrue(COSINUS.on(
        new ODouble(45)).equalsValue(new ODouble
        ↪ (0.7071067811865476))
);
}

@Test public void cosinusAngle_isCorrect3() {
    System.out.println(COSINUS.on(
        new ODouble(-45)));
    assertTrue(COSINUS.on(
        new ODouble(-45)).equalsValue(new ODouble
        ↪ (-0.7071067811865476))
);
}
}

```

Listing 65: DerivationTest (Keienburg)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import org.apache.commons.math3.analysis.polynomials.PolynomialFunction;
import org.junit.Test;

import de.fhdw.wip.rpntilecalculator.model.operands.OPolynom;
import de.fhdw.wip.rpntilecalculator.model.operands.OTuple;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

/*
 * Summary: Unit test for the derivation
 */

```

```

* Author: Jannis Luca Keienburg
* Date: 2020/01/23
*/
public class DerivationTest {
    private Derivation DERIVATION = Derivation.getInstance();

    @Test
    public void derivation_isCorrect1() {
        double[] functionValues = new double[] {2, 4, 6};
        OPolynom function = new OPolynom(new PolynomialFunction(
            ↪ functionValues));
        double[] resultValues = new double[] {4, 4};
        OPolynom derivation = new OPolynom(new PolynomialFunction(
            ↪ resultValues));
        assertEquals(DERIVATION.on(function), derivation);
    }

    @Test
    public void derivation_isCorrect2() {
        double[] functionValues = new double[] {7, 9};
        OPolynom function = new OPolynom(new PolynomialFunction(
            ↪ functionValues));
        double[] resultValues = new double[] {7};
        OPolynom derivation = new OPolynom(new PolynomialFunction(
            ↪ resultValues));
        assertEquals(DERIVATION.on(function), derivation);
    }

    @Test
    public void derivation_isCorrect3() {
        double[] functionValues = new double[] {12, -3, 12};
        OPolynom function = new OPolynom(new PolynomialFunction(
            ↪ functionValues));
        double[] resultValues = new double[] {24, -3};
        OPolynom derivation = new OPolynom(new PolynomialFunction(
            ↪ resultValues));
        assertEquals(DERIVATION.on(function), derivation);
    }

    @Test
    public void derivation_isCorrect5() {
        double[] functionValues = new double[] {1.5, 2, 7};
        OPolynom function = new OPolynom(new PolynomialFunction(
            ↪ functionValues));
        double[] resultValues = new double[] {2.25, 2};
        OPolynom derivation = new OPolynom(new PolynomialFunction(
            ↪ resultValues));
        assertEquals(DERIVATION.on(function), derivation);
    }
}

```

Listing 66: HighAndLowPointsTest (Keienburg)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import org.apache.commons.math3.analysis.polynomials.PolynomialFunction;
import org.junit.Test;

import de.fhdw.wip.rpntilecalculator.model.operands.OPolynom;
import de.fhdw.wip.rpntilecalculator.model.operands.OTuple;

import static org.junit.Assert.assertEquals;

/*
 * Summary: Unit test for the calculation of high and low points
 * Author: Jannis Luca Keienburg
 * Date: 2020/01/23
 */

public class HighAndLowPointsTest {
    private HighAndLowPoints HIGHANDLOWPOINTS = HighAndLowPoints.
        ↪ getInstance();

    @Test
    public void highAndLowPoints_isCorrect1() {
        double[] functionValues = new double[]{2, 4, 6};
        OPolynom function = new OPolynom(new PolynomialFunction(
            ↪ functionValues));
        double[] resultValues = new double[]{-1, 4};
        OTuple extremPoints = new OTuple(resultValues);
        assertEquals(HIGHANDLOWPOINTS.on(function), extremPoints);
    }

    @Test
    public void highAndLowPoints_isCorrect2() {
        double[] functionValues = new double[]{3, 6, -4};
        OPolynom function = new OPolynom(new PolynomialFunction(
            ↪ functionValues));
        double[] resultValues = new double[]{-1, -7};
        OTuple extremPoints = new OTuple(resultValues);
        assertEquals(HIGHANDLOWPOINTS.on(function), extremPoints);
    }

    @Test
    public void highAndLowPoints_isCorrect3() {
        double[] functionValues = new double[]{-2, 4, 12};
        OPolynom function = new OPolynom(new PolynomialFunction(
            ↪ functionValues));
        double[] resultValues = new double[]{1, 14};
        OTuple extremPoints = new OTuple(resultValues);
        assertEquals(HIGHANDLOWPOINTS.on(function), extremPoints);
    }
}

```

Listing 67: Logarithm10Test (Keienburg)

```
package de.fhdw.wip.rpntilecalculator.model.calculation;
```

```

import org.junit.Test;
import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import static org.junit.Assert.assertTrue;

/*
 * Summary: Unit test for the logarithm with base 10.
 * Author: Jannis Luca Keienburg
 * Date: 2020/01/22
 */

public class Logarithm10Test {
    private Logarithm10 LOGARITHM10 = Logarithm10.getInstance();

    @Test
    public void logarithm10_isCorrect1() {
        System.out.println(LOGARITHM10.on(
            new ODouble(10)));
        assertTrue(LOGARITHM10.on(
            new ODouble(10)).equalsValue(new ODouble(1.0))
    );
}

    @Test
    public void logarithm10_isCorrect2() {
        System.out.println(LOGARITHM10.on(
            new ODouble(5)));
        assertTrue(LOGARITHM10.on(
            new ODouble(5)).equalsValue(new ODouble(0.6989700043360189)
            ↪ )
    );
}

    @Test
    public void logarithm10_isCorrect3() {
        System.out.println(LOGARITHM10.on(
            new ODouble(97)));
        assertTrue(LOGARITHM10.on(
            new ODouble(97)).equalsValue(new ODouble
            ↪ (1.9867717342662448))
    );
}
}

```

Listing 68: LogarithmTest (Keienburg)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import org.junit.Test;

import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;

import static org.junit.Assert.assertTrue;

/*
 * Summary: Unit test for the natural logarithm and logatithm with

```

```

    ↗ specific base
* Author: Jannis Luca Keienburg
* Date: 2020/01/22
*/
public class LogarithmTest {
    private Logarithm LOGARITHM = Logarithm.getInstance();

    @Test
    public void naturalLogarithm_isCorrect1() {
        System.out.println(LOGARITHM.on(
            new ODouble(10)));
        assertTrue(LOGARITHM.on(
            new ODouble(10)).equalsValue(new ODouble(2.302585092994046)
            ↗
        );
    }

    @Test
    public void naturalLogarithm_isCorrect2() {
        System.out.println(LOGARITHM.on(
            new ODouble(5)));
        assertTrue(LOGARITHM.on(
            new ODouble(5)).equalsValue(new ODouble(1.6094379124341003)
            ↗
        );
    }

    @Test
    public void naturalLogarithm_isCorrect3() {
        System.out.println(LOGARITHM.on(
            new ODouble(97)));
        assertTrue(LOGARITHM.on(
            new ODouble(97)).equalsValue(new ODouble(4.574710978503383)
            ↗
        );
    }

    @Test
    public void naturalLogarithm_isCorrect4() {
        System.out.println(LOGARITHM.on(
            new ODouble(Math.exp(1))));
        assertTrue(LOGARITHM.on(
            new ODouble(Math.exp(1))).equalsValue(new ODouble(1))
        );
    }

    @Test
    public void logarithmOfChoosenBase_isCorrect1() {
        System.out.println(LOGARITHM.on(
            new ODouble(5),
            new ODouble(10)));
        assertTrue(LOGARITHM.on(
            new ODouble(5),

```

```
        new ODouble(10))
    .equalsValue(new ODouble(1.4306765580733933))
);
}

@Test
public void logarithmOfChoosenBase_isCorrect2() {
    System.out.println(LOGARITHM.on(
        new ODouble(10),
        new ODouble(10)));
    assertTrue(LOGARITHM.on(
        new ODouble(10),
        new ODouble(10))
        .equalsValue(new ODouble(1.0)))
);
}

@Test
public void logarithmOfChoosenBase_isCorrect3() {
    System.out.println(LOGARITHM.on(
        new ODouble(Math.exp(1)),
        new ODouble(10)));
    assertTrue(LOGARITHM.on(
        new ODouble(Math.exp(1)),
        new ODouble(10))
        .equalsValue(new ODouble(2.302585092994046)))
);
}
```

Listing 69: SinusTest (Keienburg)

```
package de.fhdw.wip.rptilecalculator.model.calculation;

import org.junit.Test;
import de.fhdw.wip.rptilecalculator.model.operands.ODouble;
import static org.junit.Assert.assertTrue;

/*
 * Summary: Unit test for the sinus function
 * Author: Jannis Luca Keienburg
 * Date: 2020/01/22
 */

public class SinusTest {
    private Sinus SINUS = Sinus.getInstance();

    @Test public void sinusAngle_isCorrect1() {
        System.out.println(SINUS.on(
            new ODouble(10)));
        assertTrue(SINUS.on(
            new ODouble(10)).equalsValue(new ODouble
                ↩ (0.17364817766693033))
    );
}
```

```

    @Test public void sinusAngle_isCorrect2() {
        System.out.println(SINUS.on(
            new ODouble(45)));
        assertTrue(SINUS.on(
            new ODouble(45)).equalsValue(new ODouble
                ↪ (0.7071067811865475))
    );
}

@Test public void sinusAngle_isCorrect3() {
    System.out.println(SINUS.on(
        new ODouble(-45)));
    assertTrue(SINUS.on(
        new ODouble(-45)).equalsValue(new ODouble
            ↪ (-0.7071067811865475))
}
}

```

Listing 70: TangensTest (Keienburg)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import org.junit.Test;
import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import static org.junit.Assert.assertTrue;

/*
 * Summary: Unit test for the tangens
 * Author: Jannis Luca Keienburg
 */

public class TangensTest {
    private Tangens TANGENS = Tangens.getInstance();

    @Test
    public void tangensAngle_isCorrect1() {
        System.out.println(TANGENS.on(
            new ODouble(10)));
        assertTrue(TANGENS.on(
            new ODouble(10)).equalsValue(new ODouble
                ↪ (0.17632698070846498))
    }
}

    @Test
    public void tangensAngle_isCorrect2() {
        System.out.println(TANGENS.on(
            new ODouble(45)));
        assertTrue(TANGENS.on(
            new ODouble(45)).equalsValue(new ODouble
                ↪ (0.9999999999999999)))
    }
}

```

```

    @Test
    public void tangensAngle_isCorrect3() {
        System.out.println(TANGENS.on(
            new ODouble(-45)));
        assertTrue(TANGENS.on(
            new ODouble(-45)).equalsValue(new ODouble
                ↪ (-0.9999999999999999)))
    );
}
}

```

Listing 71: ZerosTest (Keienburg)

```

package de.fhdw.wip.rpntilecalculator.model.calculation;

import org.apache.commons.math3.analysis.polynomials.PolynomialFunction;
import org.junit.Test;
import de.fhdw.wip.rpntilecalculator.model.operands.ODouble;
import de.fhdw.wip.rpntilecalculator.model.operands.OPolynom;
import de.fhdw.wip.rpntilecalculator.model.operands.OTuple;
import static org.junit.Assert.assertTrue;

/*
 * Summary: Unit test for the calculation of the zeros
 * Author: Jannis Luca Keienburg
 * Date: 2020/01/23
 */

public class ZerosTest {
    private Zeros ZEROS = Zeros.getInstance();

    @Test
    public void zeros_isCorrect1() {
        double[] functionValues = new double[] {2, 4};
        OPolynom function = new OPolynom(new PolynomialFunction(
            ↪ functionValues));
        OTuple result = new OTuple(-2);
        System.out.println(ZEROS.on(function));
        assertTrue(ZEROS.on(new OPolynom(new PolynomialFunction(
            ↪ functionValues)))
            .equalsValue(result));
    }

    @Test
    public void zeros_isCorrect2() {
        double[] functionValues = new double[] {2, 4, 0};
        OPolynom function = new OPolynom(new PolynomialFunction(
            ↪ functionValues));
        OTuple result = new OTuple(-2, 0);
        System.out.println(ZEROS.on(function));
        assertTrue(ZEROS.on(new OPolynom(new PolynomialFunction(
            ↪ functionValues)))
            .equalsValue(result));
    }
}

```

```
@Test
public void zeros_isCorrect3() {
    double[] functionValues = new double[] {1, 4, -4};
    OPolynom function = new OPolynom(new PolynomialFunction(
        ↪ functionValues));
    OTuple result = new OTuple(-4.828, 0.828);
    System.out.println(ZEROS.on(function));
    assertTrue(ZEROS.on(new OPolynom(new PolynomialFunction(
        ↪ functionValues)))
        .equalsValue(result));
}
}
```

10.2 View

10.2.1 Layout

Listing 72: ScreenOrientation

XXX

Listing 73: StorageLoadingException

XXX

Listing 74: TileLayout

XXX

Listing 75: TileLayoutFactory

XXX

Listing 76: TileLayoutLoader

XXX

10.2.2 Menu

Listing 77: ChooseListMenu

XXX

Listing 78: DialogMenu

XXX

Listing 79: InputDouble

XXX

Listing 80: InputFraction

XXX

Listing 81: InputMenuFactory

XXX

Listing 82: InputPolynomial

XXX

Listing 83: InputTileType

XXX

10.2.3 Schemas

Listing 84: ActionTileScheme

XXX

Listing 85: ErrorTileScheme

XXX

Listing 86: HistoryTileScheme

XXX

Listing 87: OperandTileScheme

XXX

Listing 88: SettingTileScheme

XXX

Listing 89: StackTileScheme

XXX

Listing 90: TileScheme

XXX

10.2.4 Sonstiges

Listing 91: Tile

XXX

Listing 92: TileMapping

XXX

Listing 93: TileType

XXX

Listing 94: TypeQuestionable

XXX

Listing 95: MainActivity

XXX

10.3 Presenter

Listing 96: Presenter

XXX

11 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, 6. Februar 2020

Max Mustermann