Studienarbeit

Reverse Polish Notation Tile Calculator

Teilprüfungsleistung in WIP

Erstellt von dem Team 1 "Das Proletariat":

Tom Bockhorn
2715438
Mülheimer Straße 274
51469 Bergisch Gladbach

Hendrik Falk
2715450
An der Josefshöhe 33
53117 Bonn

Dennis Gentges
2715460
Zum Bahnert 22
50189 Elsdorf

Getuart Istogu
2715526
Gerberstr. 3
51688 Wipperfürth

Jannis Luca Keienburg
2715548
Ruthe Furth 4
51515 Kürten

Tim Jonas Meinerzhagen
2715581
Kamper Weg 1
51519 Odenthal

Khang Pham
2715614
Vereinsstr. 15
51379 Leverkusen

Tim Schwenke
2715670
Mülheimer Straße 274
51469 Bergisch Gladbach

Prüfer: Prof. Dr. Thomas Seifert

Eingereicht am: 4. Februar 2020

Inhaltsverzeichnis

Abbildungsverzeichnis

Listingverzeichnis

1 Das Team mit Namen und Bild

Abbildung 1: Gruppenfoto

Abbildung 2: Tom Bockhorn

Abbildung 3: Hendrik Falk

Abbildung 4: Dennis Gentges

Abbildung 5: Getuart Istogu

Abbildung 6: Jannis Keienburg

Abbildung 7: Tim Meinerzhagen

Abbildung 8: Khang Pham

Abbildung 9: Tim Schwenke

2 Ziel des Projekts

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

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

3 Projektplanung

3.1 Beschreibung des Funktionsumfangs

3.2 Projektablaufplan

3.3 Planung der Software

3.3.1 Planung des Mockups

3.3.2 Planung der Datenstrukturen und Schnittstellen

Nutzung von Stack für Notation [Schwenke]

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

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

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

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

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

Ansatz der Kalkulationsorchestrierung [Schwenke]

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

Listing 1: Konzept für Nutzung generischer Schnittstelle

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

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

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

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

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

3.3.3 Planung der Activities und Layouts

3.3.4 Planung der Navigation zwischen den Activities

3.4 Geplante Aufgabenverteilung im Team (tabellarisch)

4 Beschreibung des Projektverlaufs

4.1 Tatsächliche Aufgabenverteilung im Team (tabellarisch)

4.2 Teammeeting-Protokolle

Datum	Dauer	Beschreibung
03.09.2019	200 Min.	Auswahl des Projekttyps. Entscheidung für die Entwicklung einer Android-App. Erste Einarbeitung in die Thematik. Lesen des bereitgestellten Dokuments mit Aufgabenstellung, groben Anforderungen und weiteres. Konzepterarbeitung auf Papier. Vorstellung und Diskussion verschiedener Ansätze.
04.09.2019	110 Min.	Einigung mit dem Dozenten auf einen Ansatz für den Taschenrechner. Workflow für Git, Meeting-Protokolle, Studienarbeit und Projekttagebücher festlegen. Ausarbeitung des Konzepts für den Taschenrechner. Hier wurden dem Auftraggeber Herr Seifert mehrere Konzepte vorgestellt und gemeinsam mit ihm genaue Anforderungen erarbeitet.
05.09.2019	20 Min.	Fertigstellung des Konzeptes. Fortschritte beim Paper-Prototypen. Android-Umgebung ist bei allen Team-Mitgliedern komplett aufgesetzt und lauffähig.
	60 Min.	Besprechung des aktuellen Stands des Konzepts und des Prototypen. Erstellung eines Mid-Fidelity Prototypen.

Tabelle wird auf der nächsten Seite fortgesetzt...

...Forsetzung der Tabelle

Datum	**Dauer**	**Beschreibung**
		Diskussion über Umsetzung und Workflow der App. – Wie sollen die einzelnen Kacheln funktionieren? – Wie sollen die Kacheln miteinander interagieren? – Wie könnte die Architektur der App aussehen?
	40 Min.	Projektplanung mithilfe von Projektstrukturplan und weiteren Methoden. Übertragen der Ergebnisse in den Teams-Planner. Grobe Verteilung der Arbeitspakete innerhalb des Teams.
17.09.2019	60 Min.	Erweiterung UML-Klassendiagramm. Die Klasse Operand wird abstrakt und wird von konkreten Operanden wie Vector geerbt. Diese stellen Extensions dar die neben den eigentlichen mathematischen Werten weitere Daten und Verhalten mitbringen. Welche Library soll für Mathe-Funktionalitäten benutzt werden? JScience und die bereits mitgelieferte Standardbibliothek.
17.09.2019	90 Min.	Wie sollen Elemente in der GUI dargestellt werden? Als Character oder gerendert in LaTeX. Letzteres ist mit höherer Komplexität verbunden sieht aber auch besser aus. Wie soll das Layout funktionieren? Gridlayout fällt raus, weil nicht dynamisch genug? Relative-Layout ist eine Option. Hier darf aber die Anordnung beim Rotieren nicht unkontrolliert verändert werden. UI Team möchte, dass alle Komponenten gleich groß sind. In dem Fall kann man Gridlayout benutzen.

Tabelle wird auf der nächsten Seite fortgesetzt...

...Forsetzung der Tabelle

Datum	**Dauer**	**Beschreibung**
		Wie soll die Eingabe von Funktionen im Graph Operand funktionieren? Nur möglich mit bereits vorhandenen Elementen in der Oberfläche. Es öffnet sich keine Tastatur.
09.10.2019	90 Min.	Vorstellung des Backend-Entwurfs für Teammitglieder, die für das Frontend zuständig sind. Vorstellung des Frontend-Entwurfs für Teammitglieder, die für das Backend zuständig sind. Diskussion über Verbindung von Frontend und Backend. Wie abgekoppelt lässt sich der Calculator wirklich realisieren? Vorstellung der Hauptbibliothek die für die (aufwändigen) Rechnungen wie Nullstellenberechnung benutzt werden soll. Warum Apache Commons Math und nicht JScience? Diskussion des Programm-Workflows.
05.01.2020	120 Min.	Aufnahme des aktuellen Projektstands. Besprechung des weiteren Vorgehens. Aufgabenabstimmung. Besprechung des geplanten Frontends. Besprechung/ Lösung von Problemen.
14.01.2019	90 Min.	Zusammenführung Frontend Backend Präsentation des Frontends durch das GUI-Team. Besprechen von MVC-Umsetzung in Android. Backend Unit-Testing Fortschritte.

Tabelle wird auf der nächsten Seite fortgesetzt...

...Forsetzung der Tabelle

Datum	**Dauer**	**Beschreibung**
		Serialisierung der Stacks zur Session-Sicherung.
24.01.2019	240 Min.	Detaillierte Ausarbeitung der Architektur im Backend. Programmieren im Team. Zusammenführen mehrere Features. Umbau der Programmstruktur.
28.01.2019	90 Min.	Gespräch mit Herr Prof. Dr. Thomas Seifert über den aktuellen Stand des Projekts und im Anschluss daran eine Nachbesprechung innerhalb des Teams. Vorstellung: – Vorstellung der bereits implementierten Grundfunktionen der App. – Vorstellung des verwendeten Design-Patterns. – Abgleich von Umsetzung mit den Anforderungen des Dozenten. – Ansatz des Backends erklärt. – Gerät ausleihen, um nicht nur mit Emulator testen zu können. – Serialisierung der Daten (Speichern und Laden).

Tabelle wird auf der nächsten Seite fortgesetzt...

...Forsetzung der Tabelle		
Datum	**Dauer**	**Beschreibung**
		Ergebnis: – Projekt ist auf einem guten Weg. Priorisiert werden sollen differenzierende Funktionen anstatt wenige Features sehr detailliert auszuarbeiten (Prototypische Arbeit). – Ternäre, Quaternäre usw. Operationen sind gewünscht. – Vektoren in Bestandteile lösen. – Eingabe von Matrizen. – Jeder Klasse muss ein Verantwortlicher zugeordnet sein. Ideen aus der Nachbesprechung: – ”Vektor bauen” / ”Vektoren auflösen” Action. – Summe von Stack Action. – 1x Triple Operator einfügen. – Operanden Eingabe via einzelne Menüs. – Ranks der Stacks anpassen. – Format des ersten Stacks anpassen.
03.02.2019	90 Min.	Besprechen des aktuellen Stands der App. Was muss noch unbedingt umgesetzt werden? Aufteilung der noch offenen Kapitel in der Ausarbeitung. Neues Kapitel ”Einleitung” mit Motivation. Anpassung einiger Kapitelbezeichnungen an Gegebenheiten des Projekts. Koordination der Ausarbeitung.
Summe der Dauer aller Meetings beträgt 21 Stunden		

4.3 Projekttagebücher aller Teammitglieder (tabellarisch)

4.3.1 Tom Bockhorn

4.3.2 Hendrik Falk

4.3.3 Dennis Gentges

4.3.4 Getuart Istogu

4.3.5 Jannis Keienburg

4.3.6 Tim Jonas Meinerzhagen

4.3.7 Khang Pham

4.3.8 Tim Schwenke

4.4 Beschreibung von Problemen

4.4.1 Softwareentwicklung im Team [Schwenke]

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

mit dem Remote zu synchronisieren. Mit dieser Herangehensweise verpasst man unter Umständen große Fortschritte im Gesamtprojekt. Die lokale Version ist plötzlich nicht mehr lauffähig und muss aufwändig angepasst werden. Deswegen wird im Rahmen dieses Projekts der *Gitflow-Workflow* verwendet. Grafisch dargestellt ist dieser beispielhaft in der folgenden Grafik.

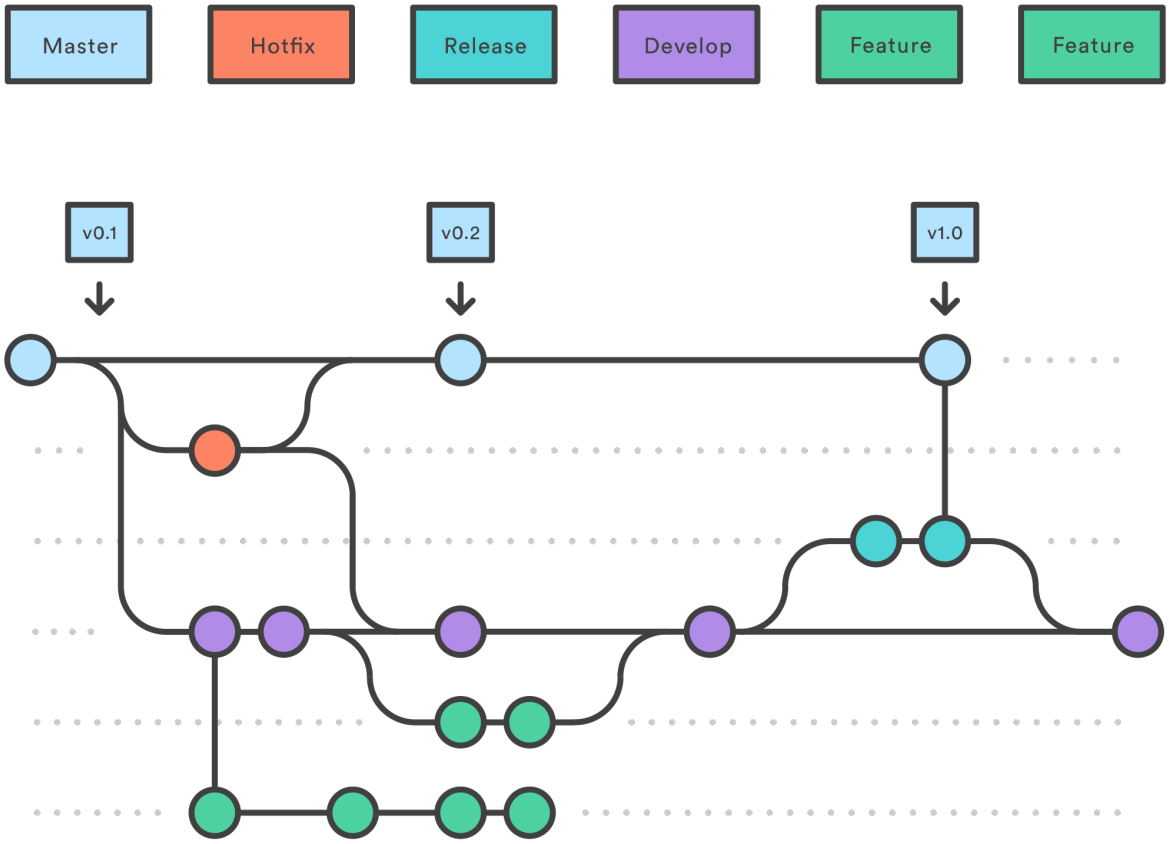

Abbildung 10: Gitflow[1]

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

`feature/backend/<konkretes-feature>`

`feature/frontend/<konkretes-feature>`

[1] Atlassian (2020)

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

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

5 Dokumentation der Software

5.1 Dokumentation der Paketstruktur des Android-Projektes

5.2 Überblick über die Activities der App bzw. der Funktionen

5.3 Dokumentation der Navigation zwischen Activities

5.4 Dokumentation der Activity-übergreifenden, persistenten Datenhaltung

5.5 Dokumentation der programmatischen Beiträge der Teammitglieder

5.5.1 Tom Bockhorn

5.5.2 Hendrik Falk

5.5.3 Dennis Gentges

5.5.4 Getuart Istogu

5.5.5 Jannis Keienburg

5.5.6 Tim Jonas Meinerzhagen

5.5.7 Khang Pham

5.5.8 Tim Schwenke

Generische Kalkulationsorchestrierung

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

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

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

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

Listing 2: Methodenkopf der generischen Schnittstelle

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

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

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

Listing 3: Implementierung der generischen Schnittstelle

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

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

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

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

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

6 Dokumentation der sonstigen Beiträge der Teammitglieder

6.1 Tom Bockhorn

6.2 Hendrik Falk

6.3 Dennis Gentges

6.4 Getuart Istogu

6.5 Jannis Keienburg

6.6 Tim Jonas Meinerzhagen

6.7 Khang Pham

6.8 Tim Schwenke

7 Fazits aller Teammitglieder

7.1 Tom Bockhorn

7.2 Hendrik Falk

7.3 Dennis Gentges

7.4 Getuart Istogu

7.5 Jannis Keienburg

7.6 Tim Jonas Meinerzhagen

7.7 Khang Pham

7.8 Tim Schwenke

8 Quellenverzeichnis

iju

9 Anhang - Quelltext

9.1 Model

9.1.1 Calculation

Listing 4: Action (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
 * Date:   2020/01/04
 */
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 5: 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
 * Date:   2020/01/04
 */

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 6: 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
 * Date:   2020/01/04
 */
```

```
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 7: 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
 * Date:   2020/01/04
 */

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 8: CalculationException (Schwenke)

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

/**
 * Summary: Main Exception for a failed Calculation
 * Author: Tim Schwenke
 * Date:   2020/01/04
 */
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 9: 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;
```

```java
/*
 * Summary: Defines the Cosinus action.
 * Author: Jannis Luca Keienburg
 * Date:   2020/01/04
 */
@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 10: Derivation (Keienburg)

```java
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
 * Date:   2020/01/23, updated 2020/02/02
 */

public class Derivation extends Action {

    @NotNull public static final Derivation DERIVATION = new Derivation();
```

```
    @Contract(pure = true) @NotNull public static Derivation getInstance()
        ↪  { return DERIVATION; }
    public 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 11: 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
 * Date:   2020/01/23 (initialy) updates at the 02/02/20
 */

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 uneyual 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 12: 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
 * Date:   2020/01/27
 */

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[] functionCoefficents = polynomialFunction.getCoefficients()
        ↪ ;

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

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

```
                ↪ double) i);
        }

        return new OPolynom(new PolynomialFunction(
            ↪ antiderivativeCoefficents));
    }
}
```

Listing 13: 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
 * Date:   2020/01/04
 */
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 < 0.00000000001) {
                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 < 0.00000000001) {
                    d = approach;
                }
            }
        }
        return Double.NaN;
    }
}
```

Listing 14: 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
 * Date:   2020/01/19
 */
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) {
        return new ODouble(Math.log(oDouble.getDouble()));
    }
```

```
    //Logarithm with specific base
    @Contract(pure = true) @NotNull ODouble on(@NotNull ODouble base,
        ↪ ODouble logartihmOf) {
        return new ODouble(Math.log(logartihmOf.getDouble()) / Math.log(
            ↪ base.getDouble()));
    }
}
```

Listing 15: 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.OSet;
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
 * Date:   2020/01/27
 */

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 OMatrix A, OTuple
        ↪ b) {
        return solveLinearSystem(A, b.getTuple());
    }
    /**
     * On the condition that A*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*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 16: Minus (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.jetbrains.annotations.Contract;
import org.jetbrains.annotations.NotNull;

/*
 * Summary: Defines the Minus click. Lets the user subtract operands.
 * Author: Tim Schwenke
 * Date:   2020/01/04
 */
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 17: Modulo (Istogu)

```
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: Getuart Istogu
 * Date:   2020/01/05
 */
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 18: 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
 * Date:   2020/01/04
 */
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 19: Power (Istogu)

```
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: Getuart Istogu
 * Date:   2020/01/04
```

```
 */

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 20: 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
 * Date:   2020/01/04
 */
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 21: Slash (Schwenke)

```
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: Tim Schwenke
 * Date:   2020/01/04
 */
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 22: 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
 * Date:   2020/01/04
 */
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 23: Times (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 Times click. Lets the user Multiplies operands.
 * Author: Tim Schwenke
 * Date:   2020/01/04
 */
@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 24: 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
 * Date:   2020/01/19, updated on 02.02.2020
 */
public class Zeros extends Action {

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

    @Contract(pure = true) @NotNull public static Zeros getInstance() {
        ↪ return ZEROS; }
    public 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 funtion
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 25: Root (Istogu)

```
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: Getuart Istogu
 * Date:   2020/01/04
 */

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()));
    }
}
```

9.1.2 Operands

Listing 26: 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
 * Date:   2020/01/08
 */
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 27: 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
 * Date:   2020/01/04
 */
public final class DoubleFormatter {

    // Decimal Format
    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 28: Element (Schwenke)

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

/*
```

```
 * Summary: Main Element Wrapper for all kinds
 * Author: Tim Schwenke
 * Date:   2020/01/04
 */
public abstract class Element {

}
```

Listing 29: 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
 * Date:   2020/01/04
 */
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 30: 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
 * Date:   2020/01/04
 */
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 31: 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
 * Date:   2020/01/04
 */
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 32: 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
 * Date:   2020/01/04
 */
@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 33: 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;
```

```java
/*
 * Summary: Wrapper for the Polynom Operand
 * Author: Tim Schwenke
 * Date:   2020/01/04
 */
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\\^[0-9])( \\+)*");
        double[] coefficients = new double[vars.length];
        for(int i = 0; i < vars.length; i++) coefficients[i] = Double.
            ↪ valueOf(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 34: 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
 * Date:   2020/01/04
 */
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 35: 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
 * Date:   2020/01/04
 */
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 36: 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
 * Date:   2020/01/04
 */
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);

}
```

9.1.3 Settings

Listing 37: 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
```

```
 * Date:   2020/01/26
 */
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 38: 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
 * Date:   2020/01/26
 */
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 39: 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
 * Date:   2020/01/26
 */
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 40: 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
 * Date:   2020/02/03
 */
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 41: 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
 * Date:   2020/01/27
 */
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 42: 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
```

```
 * Date:   2020/01/29
 */
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 43: 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
 * Date:   2020/01/26
 */
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 44: 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
```

```
 * Date:   2020/01/26
 */
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 45: Setting (Falk)

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

import org.jetbrains.annotations.Contract;

/**
 * Summary: Super class for settings
 * Author: Hendrik Falk
 * Date:   2020/01/26
 */
```

```
public abstract class Setting {

    @Contract(pure = true)
    public abstract boolean call();
}
```

Listing 46: 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.OSet;
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 ODoubles
 * Author: Hendrik Falk
 * Date:   2020/02/03
 */
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 47: 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
 * Date:   2020/01/30
 */
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 48: 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
 * Date:   2020/01/30
 */
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;
    }
}
```

9.1.4 Stack

Listing 49: StackInterface (Schwenke)

```
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: Tim Schwenke
 * Date:   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 50: OperandStack (Schwenke)

```
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: Tim Schwenke
 * Date:   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));
    }

}
```

9.2 View

9.2.1 Layout

Listing 51: ScreenOrientation

```
XXX
```

Listing 52: StorageLoadingException

```
XXX
```

Listing 53: TileLayout

```
XXX
```

Listing 54: TileLayoutFactory

```
XXX
```

Listing 55: TileLayoutLoader

```
XXX
```

9.2.2 Menu

Listing 56: ChooseListMenu

```
XXX
```

Listing 57: DialogMenu

```
XXX
```

Listing 58: InputDouble

```
XXX
```

Listing 59: InputFraction

```
XXX
```

Listing 60: InputMenuFactory

```
XXX
```

Listing 61: InputPolynomial

```
XXX
```

Listing 62: InputTileType

```
XXX
```

9.2.3 Schemas

Listing 63: ActionTileScheme

```
XXX
```

Listing 64: ErrorTileScheme

```
XXX
```

Listing 65: HistoryTileScheme

```
XXX
```

Listing 66: OperandTileScheme

```
XXX
```

Listing 67: SettingTileScheme

```
XXX
```

Listing 68: StackTileScheme

```
XXX
```

Listing 69: TileScheme

```
XXX
```

9.2.4 Sonstiges

Listing 70: Tile

```
XXX
```

Listing 71: TileMapping

```
XXX
```

Listing 72: TileType

```
XXX
```

Listing 73: TypeQuestionable

```
XXX
```

Listing 74: MainActivity

```
XXX
```

9.3 Presenter

Listing 75: Presenter

```
XXX
```

10 Anhang - Verwendeten Tools und Hilfsprogramme

kio

Ehrenwörtliche Erklärung

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

Bergisch Gladbach, 4. Februar 2020

Max Mustermann