Einfacher Plotter, Teil 3: Umgekehrte Polnische Notation

In diesem Aufgabenteil geht es darum, aus einer Funktion, die in Umgekehrter Polnischer Notation gegeben ist, einen Plot zu erstellen. Dazu muss verstanden werden, wie eine Funktion in dieser Form aussieht und wie sie sich verhält.

Falls du irgendwo Probleme hast, zögere bitte nicht, um Hilfe zu fragen! Wenn du irgendwo hängst, helfen wir dir gerne weiter! Melde dich bei einem der zuständigen Betreuer*innen: Lorenz Braun, Fabian Sand, Robin Fritz, Furkan Keserci, Julia Kisela. Wir sind über Discord, Mattermost und E-Mail erreichbar. Wen könnt ihr für Hilfe ansprechen? (E-Mail, Mattermost, Discord)

- **Robin Fritz: frro1020@h-ka.de, @robin.fritz, Snobo#4754**
- **Julia Kisela: kiju1012@h-ka.de, @julia.kisela, lohikäärme#3336**
- **Furkan Keserci: kefu1011@h-ka.de, @furkan, Killnexx#9999**
- **Lorenz Braun: brlo1014@h-ka.de, @lorenzbraun, MrRaptorious#7632**
- **Fabian Sand: safa1017@h-ka.de, @safa1017, Fabian S#7613**

Lernmaterial

In diesem Abschnitt kommen zu den Expressions (evaluierbare Ausdrücke) noch Tokens hinzu. Ein Symbol, oder mehrere zusammenhängende Symbole, ergeben einen Token. Z. B. ist '-3,5' ein Token, oder auch '+'.

Hier das Beispiel $f(x) = sin(5 * x - (3 + 8))$ und wie es sich in Umgekehrter Polnischer Notation verhält:

```
8, 3, +, x, 5,*, -, sin
```

So würde das Beispiel in Code-Form aussehen:

```
Expression e = Expression.parseRPN(new Token[]{
                new NumberToken(8),
                new NumberToken(3),
                new Token(TokenType.PLUS),
                new Token(TokenType.X),
                new NumberToken(5),
                new Token(TokenType.TIMES),
                new Token(TokenType.MINUS),
                new FunctionToken("sin")
        });
```

Listing 2: RPN Beispiel in Code-Form

Wie man genauer eine Umgekehrte Polnische Notation liest und auswertet (also in eine AST Form bringt), kannst du hier nachlesen:

https://de.wikipedia.org/wiki/Umgekehrte_polnische_Notation

Oder im Kapitel 4.3 External Variables auf Seite 67/68 zur reverse polish notiation in The C Programming Language von Kernighan und Ritchie.

https://github.com/auspbro/ebook-c/blob/master/The.C.Programming.Language.2Nd.Ed%20Prentice.Hall.Brian.W.Kernighan.and.Dennis.M.Ritchie..pdf

Dort befindet sich auch ein kleines Programm, welches einen Taschenrechner in Umgekehrter Polnischer Notation implementiert. Diesen Code nachzuvollziehen kann dir später beim Programmieren helfen.

Verständnisfragen

- Was ist der Vorteil von Umgekehrter Polnischer Notation?
- Schreibe den Ausdruck in der Infix-Schreibweise $5 * x + (log(x) - (3 + 5))$ in Umgekehrte Polnische Notation um. Orientiere dich dabei an dem Beispiel weiter oben. Du solltest auch erklären können, wie du darauf gekommen bist.
- Im letzten Aufgabenteil hast du Ausdrücke als Syntaxbaum geschrieben. Schreibe nun diesen Ausdruck in RPN

  ```
  1 2 / x *
  ```

  in einen Syntaxbaum um. Was passiert dabei? Du solltest erklären können, wie du darauf gekommen bist.
- Was ist der Unterschied zwischen einem Token und einer Expression? Gib ein Beispiel für einen Token an, der keine Expression ist.
- Was passiert, wenn man zu viele Operanden eingibt? (Hier könnte im Code eine Exception stehen.)

Aufgabenstellung

Wichtiger Hinweis: Im bereitgestellten Code finden sich Dokumentationskommentare, welche Aufschluss auf die Funktionsweise oder die zu implementierende Funktionalität geben. Bitte deshalb den Code und die Kommentare aufmerksam studieren! Bitte verändere gegebene Klassennamen und Methodennamen nicht, da diese für die Tests verwendet werden!

Nun werden nicht mehr direkt die Syntaxbaumdarstellungen zum Plotten genutzt, sondern ein Array aus Tokens in Umgekehrter Polnischer Notation. Dazu muss es einige Token-Klassen geben, um diese anlegen zu können.

Lade dazu das Paket tokens und importiere es in dein Projekt (in den src-Ordner hineinlegen). Die Klasse Expressions kannst du durch die neue ersetzen. Schreibe nun für die Klassen NumberExpression, OperatorExpression, UnaryFunctionExpression jeweils einen neuen Konstruktor, der einen passenden Token als Parameter entgegennimmt. Schreibe für die Klasse Token und ihre Tochter-Klassen FunctionToken und NumberToken Konstruktoren und Getter. Du findest TODO-Kommentare an allen Stellen, die du füllen sollst. Fülle

auch TokenType aus. Du bekommst auch einen neuen Testordner von uns, um deinen Code zu testen, ersetze den alten durch den neuen. (Die bisherigen Testfälle wird es trotzdem weiterhin geben, aber zusätzlich kommen neue hinzu.)

Um ein Array aus Tokens in Umgekehrter Polnischer Notation in eine AST-Darstellung umzuwandeln, muss ein Algorithmus entwickelt werden. Wie dieser funktioniert, hast du oben in einem Beispiel gesehen. Falls es dir noch nicht klar ist, lies es am besten im The C Programming Language Buch nach. Die Methodensignatur und ein beschreibender Kommentar befinden sich schon in der Klasse Expression.

In der Grid-Klasse müssen nun an die auszuführende Stelle Tokens in Umgekehrter Polnischer Notation geschrieben werden. Ein Beispiel sieht so aus:

```
Expression e = Expression.parseRPN(new Token[]{
        new Token(TokenType.X),
        new FunctionToken("exp"),
        new NumberToken(-1),
        new Token(TokenType.TIMES)
});
```

Füge bitte die Funktionen $sin(x)$, $log(5 * (3 + x))$, $x^2 + 3$ in der richtigen Schreibweise in deine Grid-Klasse.

Checkliste

- Konstruktor mit Token in NumberExpression
- Konstruktor mit Token in OperatorExpression
- Konstruktor mit Token in UnaryFunctionExpression
- Zwei Konstruktoren für Token
- Getter für TokenType in Token
- TokenType statt OperatorType
- Konstruktor und Getter für FunctionToken
- Konstruktor und Getter für NumberToken
- TokenType bearbeitet
- parseRPN in Expression mit Switch-Case
- Wirf eine IllegalArgumentException in parseRPN
- Fange eine EmptyStackException und wirf eine IllegalArgumentException in parseRPN
- Beispiele lassen sich plotten und sehen richtig aus