Projektabgabe Funktionsplotter

Bltte bewerten Sie den gesamten Projekt und bitte machen Sie alles geltend

Hier sind einige Erklärungen zur Projektstruktur

```
C:.
│   BinaryExpr.java
│   demo.java
│   DotExporter.java
│   Evaluator.java
│   Expr.java
│   FunctionCallExpr.java
│   InfixParser.java
│   InfixPrinter.java
│   lvp-0.5.3.jar
│   Neuer Ordner 3.iml
│   Neuer Ordner.iml
│   NumberExpr.java
│   Operator.java
│   PlotCalculator.java
│   PrefixParser.java
│   SvgPlotter.java
│   Token.java
│   Tokenizer.java
│   TokenType.java
│   UnaryExpr.java
│   UPNPrinter.java
│   VariableExpr.java
```

- Tokenizer soll den Eingabe-String in Tokens zerlegen in Number, Variable, Operator, Paren, Identifier etc.
    - Bemerkung : EOF steht für Ende
    - PLUS für + , MINUS für - , MUL für * , DIV für / , POW für ^, LPAREN für (, RPAREN für).
- Parser wird hier geteilt in InfixParser für die Konventionelle Notation mit Präsedenz und Klammern, Fehler beim Parsing wird ein Exception geworfen. Nutzt den Shunting -Yard-Algorithmus.
    - -> Damit ist sowohl die Eingabe in Infix-Noation als auch in UPN-Noation möglich für die Funktion (x+2) als Eingabe in Infix-Notation als auch 2 x + , zeichnet der Plotter dieselbe Funktion .
    - Jede Parser gib bei Erfolge einen Expr-AST zurück, bei Fehler
- AST-Modell: dafür ist der sealed interface Expr mit seinen records implementierungen wie BinaryExpr, FunctionCallExpr, NumberExpr, UnaryExpr, VariableExpr verantwolich.
- Infix Printer erstellt aus dem AST einen korrekt geklammerten Infix-String

- UPN-Printer: gibt die Postfix-Notation aus
- DotExporter.toDot(Expr) -> erzegut die Grraphische Dot Repräsentation
- Evaluator macht eine Traversierung(walk) über den AST und wertet rekursiv die Evaluator.eval() und unterstützt dabei die Funktionen wie sin, cos, log ,pow(^) ,tan abs, log etc.
- Der PlotCalculator liefert eine Liste von Punkten der Form (x,f(x))-Paaren zurück
- SvgPlotter erzeugt das SVG und wird mitgebunden and das LVP . in einem Intervall zwischen -10 und +10 horizontal und vertikal .
- Erste Funktion Wird by default mit Rot geplotet , zweite Funktion mit Grün , 3. Funktion mit Blau , 4. Funktion mit Orange und 5. Funktion in Lila .

Für jede Funktionseingabe werden die Tokens die Infix-Notation und die UPN-Notation angezeigt.

zur Eingabe der Funktionen insbesondere der Logarithmus funktion sollte es so sein log(2,x) oder log(10,x) und ln(x) sein, log(x) alleine wird es nicht funktionieren.

zur Eingabe der Betragsfunktion, es kann sowohl |x| oder mit abs(x) gehen

zur Eingabe der Wurzelfunktion mit sqrt(x)

Bitte eine Eingabe und dann send und nicht alle gleichzeitig

Eingabe ist auch möglich mit einem unären Minus, mathematische Konstanten wie e=2.71 oder Π =3.14, auch verwendung von einem Ausdruck ohne eine Variable ist möglich.

Die Eingabe von bis zu 5 Funktionen die farblich voneinander unterschliedlich sind, ist möglich.

Paar Illustrationen von dem Funktionsplotter

1.

Nach Eingabe und Klick auf **Send** werden die markierten Zeilen ersetzt.

Input Var1	5^x+1	Send
Input Var2	log(10,x)	Send
Input Var3	x^2	Send
Input Var4	1	Send
Input Var5	-(x+2)	Send

Funktion 1: `5^x+1`
Funktion 2: `log(10,x)`
Funktion 3: `x^2`
Funktion 4: `1`
Funktion 5: `-(x+2)`

Tokens 1: NUMBER:`5` · POW:`^` · NAME:`x` · PLUS:`+` · NUMBER:`1` · EOF:``

Tokens 2: NAME:`log` · LPAREN:`(` · NUMBER:`10` · COMMA:`,` · NAME:`x` · RPAREN:`)` · EOF:``

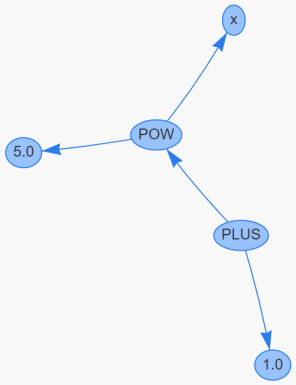
Tokens 3: NAME:`x` · POW:`^` · NUMBER:`2` · EOF:``

Tokens 4: NUMBER:`1` · EOF:``

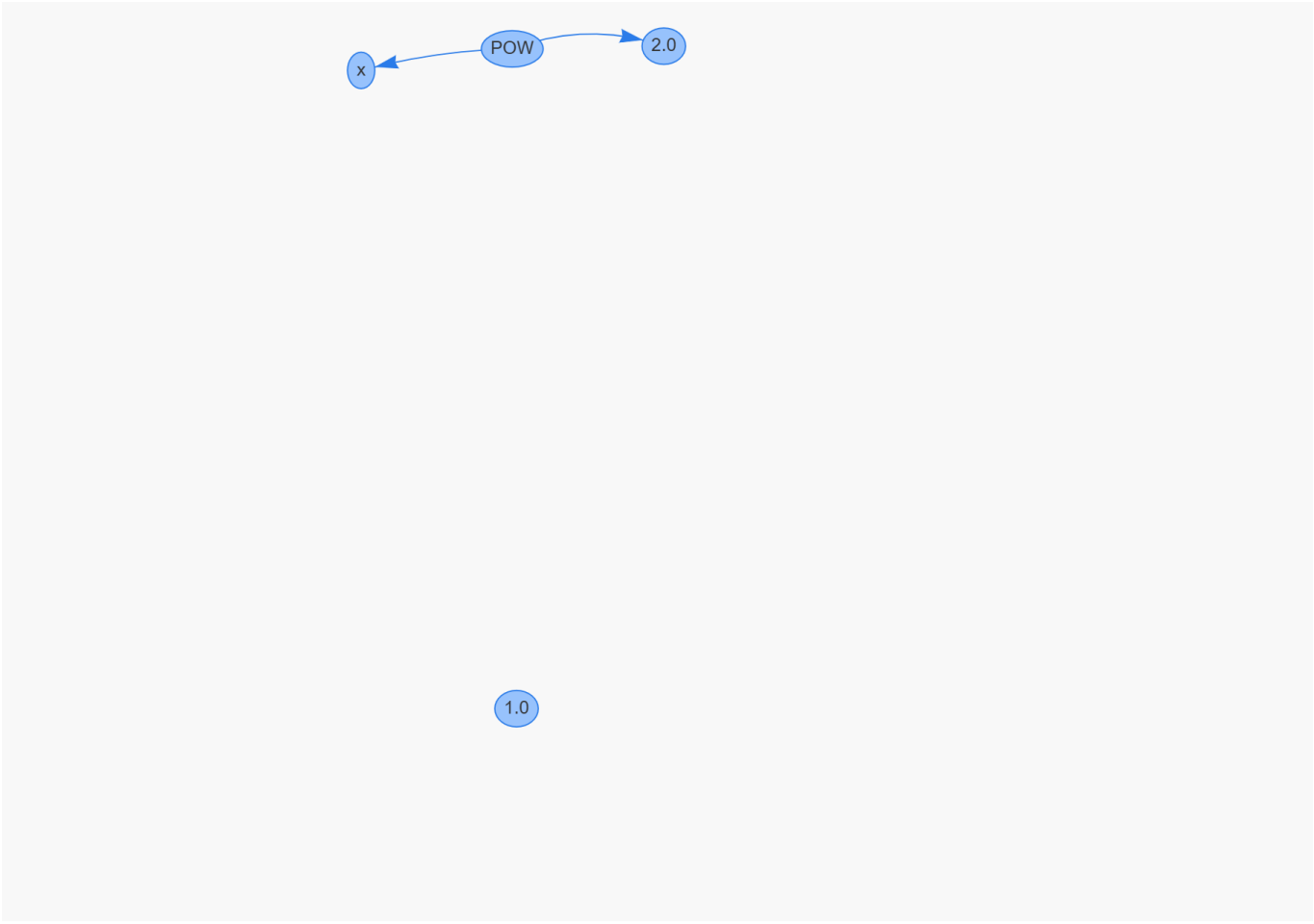
Tokens 5: MINUS:`-` · LPAREN:`(` · NAME:`x` · PLUS:`+` · NUMBER:`2` · RPAREN:`)` · EOF:``

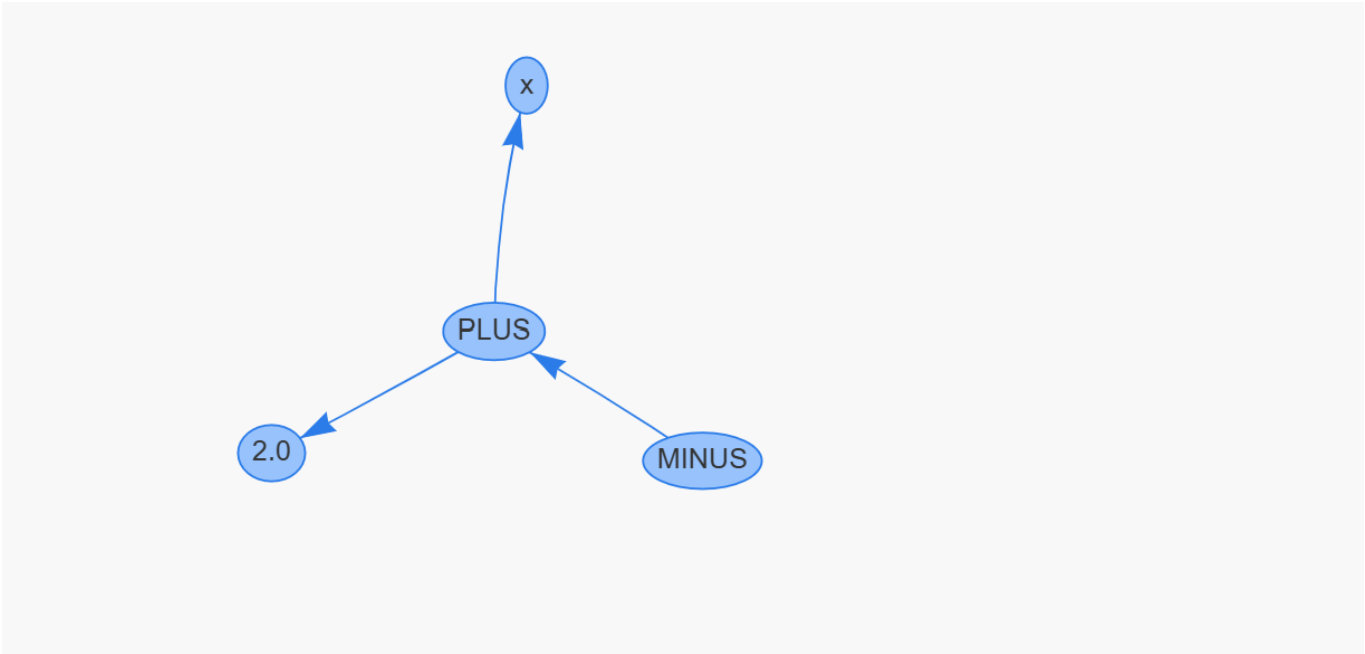
- **Infix 1:** `((5.0 ^ x) + 1.0)` | **UPN 1:** `5.0 x POW 1.0 PLUS`
- **Infix 2:** `log(10.0,x)` | **UPN 2:** `10.0 x log`
- **Infix 3:** `(x ^ 2.0)` | **UPN 3:** `x 2.0 POW`
- **Infix 4:** `1.0` | **UPN 4:** `1.0`
- **Infix 5:** `(MINUS(x + 2.0))` | **UPN 5:** `x 2.0 PLUS MINUS`

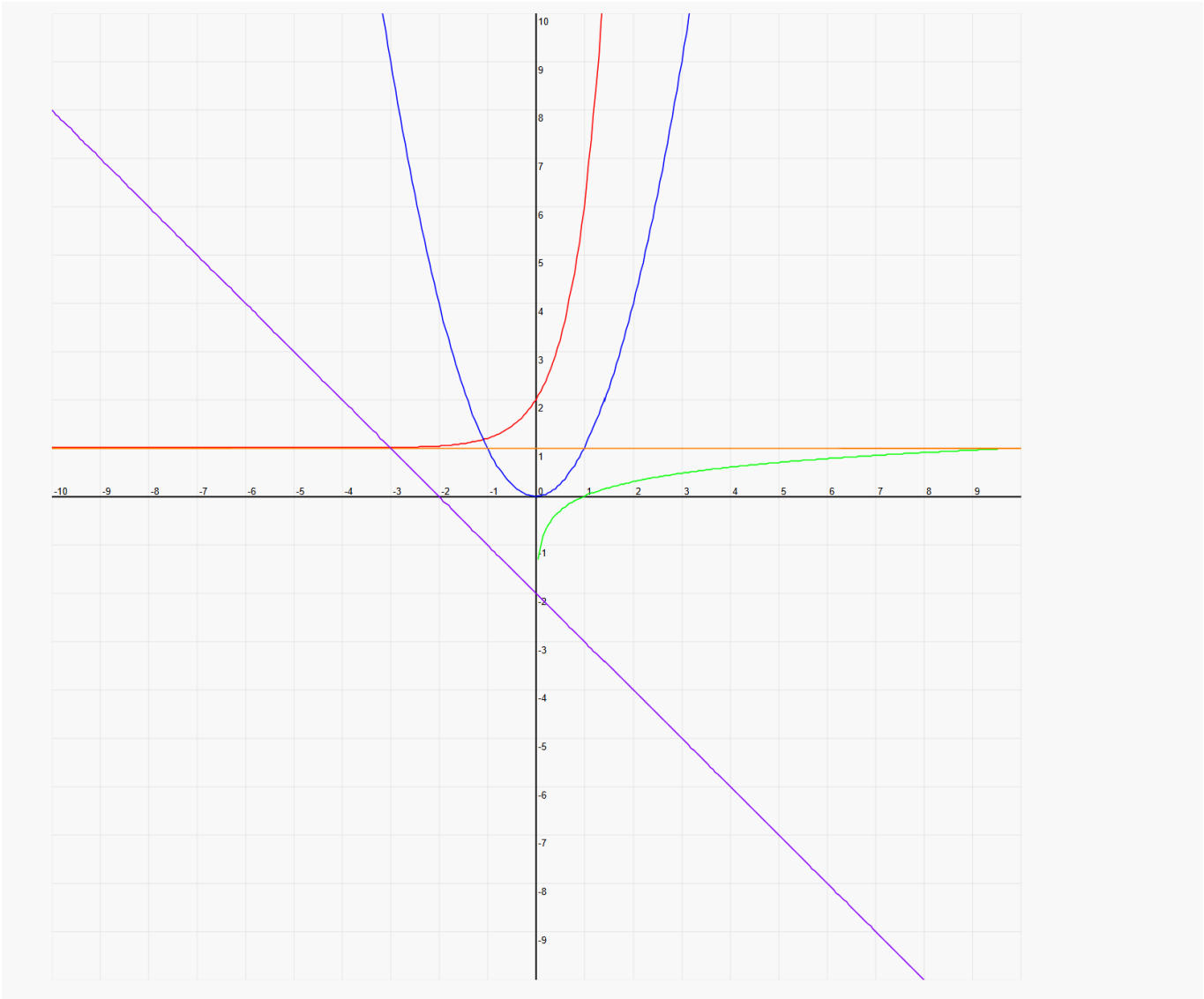
2.



3.







Sie können das Projekt unter Mein GitHub sehen:

GitHub Repository