Domáca úloha č. 1

Cieľom tejto úlohy bude naprogramovať triedy pre niektoré diferencovateľné funkcie. Naprogramujete triedy pre vybrané elementárne funkcie, ktoré budú dediť od spoločnej abstraktnej triedy `Function`. Funkcie vytvorené zložením týchto elementárnych funkcií budú reprezentované stromom podobným aritmetickému stromu.

Vytvorte balík `functions` a v ňom abstraktnú triedu `Function`, ktorá bude mať nasledovné verejné abstraktné metódy:

- `double evaluate(double x)`: metóda, ktorá vyhodnotí danú funkciu v bode `x`. Na vyhodnocovanie môžete použiť metódy triedy `Math`.
- `Function derivative()`: metóda, ktorá vráti funkciu, ktorá je deriváciou tejto. Pravidlá derivácie sú nižšie.
- `Function simplify()`: metóda, ktorá vráti funkciu zjednodušenú podľa pravidiel rozpísaných nižšie.
- okrem týchto verejných metód budú funkcie prekrývať metódu `toString()`.

Nulárne funkcie

Priamo od `Function` dedia nasledovné funkcie:

- `Constant` reprezentujúca konštantnú funkciu s konštruktorom `public Constant(double value)`, kde `value` je jej hodnota. Metóda `toString` vráti hodnotu zaokrúhlenú na 5 desatinných miest, pričom koncové nuly za desatinnou bodkou sa nebudú zobrazovať. Navyše bude poskytovať verejnú metódu `double getValue()` vracajúcu svoju hodnotu.

  Poznámka: Na zaokrúhľovanie môžete použiť triedu `DecimalFormat`. V ďalších triedach, kde sa bude hodnota typu `double` konvertovať na `string` sa bude zaokrúhľovať rovnakým spôsobom.
- `Variable`, reprezentujúca premennú x, t.j. identickú funkciu. `toString` vráti `"x"`.

Unárne funkcie

Vytvorte abstraktnú triedu `UnaryFunction`, ktorá bude dediť of `Function`. V konštruktore dostane funkciu a bude poskytovať verejnú metódu `Function getChild()`, ktorá vráti túto funkciu.
Nasledovné triedy budú dediť od `UnaryFunction`:

- `UnaryMinus` reprezentujúca unárne mínus. Ak funkcia v konštruktore reprezentuje funkciu $f(x)$, táto trieda reprezentuje funkciu $-f(x)$. Pre inštanciu s dieťatom `Variable` vráti `toString()` reťazec `"-x"`.
- `Sine` reprezentujúca sínus. Ak funkcia v konštruktore reprezentuje funkciu $f(x)$, táto trieda reprezentuje funkciu $\sin(f(x))$. Pre inštanciu s dieťatom `Variable` vráti `toString()` reťazec `"sin(x)"`.
- `Cosine` reprezentujúca kosínus. Ak funkcia v konštruktore reprezentuje funkciu $f(x)$, táto trieda reprezentuje funkciu $\cos(f(x))$. Pre inštanciu s dieťatom `Variable` vráti `toString()` reťazec `"cos(x)"`.
- `Power` reprezentujúca mocninovú funkciu s konštruktorom
  `Power(Function base, double exponent)`. Ak `base` reprezentuje funkciu $f(x)$ a `exponent` je a, táto trieda reprezentuje funkciu $f(x)^a$. Pre inštanciu so základom `Variable` a exponentom 2 vráti `toString()` reťazec `"x^2"`. Navyše bude poskytovať verejné metódy `Function getBase()`, vracajúcu základ, a `double getExponent()`, vracajúcu exponent.
- `Exponential` reprezentujúca mocninovú funkciu s konštruktorom
  `Exponential(double base, Function exponent)`. Ak `base` je a a `exponent` reprezentuje funkciu $f(x)$, táto trieda reprezentuje funkciu $a^{f(x)}$. Pre inštanciu so základom 2 a exponentom `Variable` vráti `toString()` reťazec `"2^x"`. Navyše bude poskytovať verejné metódy `double getBase()`, vracajúcu základ, a `Function getExponent()`, vracajúcu exponent.

- `Logarithm` reprezentujúca mocninovú funkciu s konštruktorom `Logarithm(double base, Function child)`. Ak `base` je a a `child` reprezentuje funkciu $f(x)$, táto trieda reprezentuje funkciu $\log_a f(x)$. Pre inštanciu so základom 2 a dieťaťom `Variable` vráti `toString()` reťazec `"log_2(x)"`. Špeciálne sa `toString` správa, ak je základ rovný `Math.E`, reprezentujúci číslo e, vtedy vráti `"ln(x)"`. Navyše bude poskytovať verejnú metódu `double getBase()`, vracajúcu základ.

Exponenciálnu funkciu a logaritmus budeme definovať len ak je základ kladný a nie rovný 1. Táto podmienka sa skontroluje v konštruktore príslušných funkcií a ak neplatí vyhodí sa výnimka:
Do balíka naporgramujte výnimku `IllegalBaseException` dediacej od `RuntimeException`, ktorú vyhodí konštruktor logaritmu a exponenciálnej funkcie. Prekryte metódu `getMessage`, ktorá vráti správu `"<function> cannot have <value> as base."`, kde `<function>` je `"logarithm"` alebo `"exponential function"`, podľa toho, ktorý konštruktor vyhodil výnimku, a `<value>` je základ ktorý konštruktor dostal ako argument.

Binárne funkcie

Vytvorte abstraktnú triedu `BinaryFunction`, ktorá bude dediť of `Function`. V konštruktore dostane dve funkcie `left` a `right` a bude poskytovať verejné metódy `Function getLeft()` a `Function getRight()`, ktoré vrátia tieto funkcie.
Nasledovné triedy budú dediť od `BinaryFunction`:

- `Plus` reprezentujúca sčítanie. Ak funkcie v konštruktore reprezentujú funkcie $f(x)$ a $g(x)$, táto trieda reprezentuje funkciu $f(x) + g(x)$. Pre inštanciu s deťmi `Variable` a konštantou 2 vráti `toString()` reťazec `"x+2"`.

- `Minus` reprezentujúca odčítanie. Ak funkcie v konštruktore reprezentujú funkcie $f(x)$ a $g(x)$, táto trieda reprezentuje funkciu $f(x) - g(x)$. Pre inštanciu s deťmi `Variable` a konštantou 2 vráti `toString()` reťazec `"x-2"`.

- `Multiplication` reprezentujúca násobenie. Ak funkcie v konštruktore reprezentujú funkcie $f(x)$ a $g(x)$, táto trieda reprezentuje funkciu $f(x)g(x)$. Pre inštanciu s deťmi `Variable` a konštantou 2 vráti `toString()` reťazec `"x*2"`.

- `Division` reprezentujúca delenie. Ak funkcie v konštruktore reprezentujú funkcie $f(x)$ a $g(x)$, táto trieda reprezentuje funkciu $\frac{f(x)}{g(x)}$. Pre inštanciu s deťmi `Variable` a konštantou 2 vráti `toString()` reťazec `"x/2"`.

Uzátvorkovanie reťazcov

Ak trieda s operátorom +, −, ⋆, / a ^ (t.j. triedy `Plus`, `Minus`, `UnaryMinus`, `Multiplication`, `Division`, `Power` a `Exponential`) má ako dieťa tiež triedu s operátorom, pri tvorbe reťazcov v metóde `toString` sa vnútorná trieda obalí do zátvoriek. Napríklad: `x+(1/x)`, `sin(x)*(-x)`, `(ln(x)+1)^2`.

Zjednodušovanie

Pravidlá pre metódu `simplify`. Ak funkcia závisí len od konštánt a nie od premennej x, metóda `simplify` vypočíta výraz, ktorý táto funkcia reprezentuje a vráti jeho hodnotu ako konštantnú funkciu.

neutrálne a absorpčné prvky: Ak funkcia pre sčítanie má po zjednodušení jej detí ako jedno dieťa konštantnú nulu ($0 + f$ alebo $f + 0$), nahradí sa druhým dieťaťom ($\to f$). Podobne aj nasledujúce pravidlá:
$f - 0 \to f$, $0 - f \to -f$,
$1 * f \to f$, $f * 1 \to f$, $f/1 \to f$,
$0 * f \to 0$, $f * 0 \to 0$, $0/f \to 0$,
$f^0 \to 1$, $f^1 \to f$.

sčítanie a odčítanie: Ak funkcia pre sčítanie má po zjednodušení jej detí ako druhé dieťa unárne mínus $(f + (-g))$, nahradí sa odčítaním/binárnym mínus ľavého dieťaťa a dieťaťa pravého dieťaťa. $(\rightarrow f - g)$. Podobne aj nasledujúce pravidlá:
$(-f) + g \rightarrow g - f$, $(-f) + (-g) \rightarrow -(f + g)$,
$f - (-g) \rightarrow f + g$, $(-f) - g \rightarrow -(f + g)$, $(-f) - (-g) \rightarrow g - f$,
$-(-f) \rightarrow f$, $-(f - g) \rightarrow g - f$.

Derivovanie

$$
\begin{aligned}
a' &= 0 \quad (a \in \mathbb{R}) \\
(f \pm g)' &= f' \pm g' \\
(f \cdot g)' &= f' \cdot g + f \cdot g' \\
\left(\frac{f}{g}\right)' &= \frac{f'g + fg'}{g^2} \\
(x^a)' &= a \cdot x^{a-1} \\
(\sin x)' &= \cos x \\
(\cos x)' &= -\sin x \\
(a^x)' &= a^x \cdot \ln a \\
(\log_a x)' &= \frac{1}{x \cdot \ln a} \\
f(g(x))' &= f'(g(x)) \cdot g'(x)
\end{aligned}
$$

Porovnávanie funkcií

Funkcie sami o sebe nevieme porovnávať a usporiadať. Vieme však porovnať hodnoty funkcií v nejakom bode. Pre diferencovateľné funkcie vieme porovnávať aj rast funkcie v nejakom bode, teda hodnotu ich derivácie v tom bode.
Naprogramujte dva komparátory implementujúce rozhranie `Comparator<Function>`:

- `ComparingByValue` s konštruktorom `ComparingByValue(double point)`, ktorý bude porovnávať funkcie podľa ich hodnoty v bode `point` a
- `ComparingByRateOfChange` s konštruktorom `ComparingByRateOfChange(double point)`, ktorý bude porovnávať funkcie podľa ich rastu v bode `point`.

Vo triede `Function` naprogramujte verejnú statickú metódu `void sortFunctions(List<Function> list, double point)`, ktorá utriedi a vypíše list funkcií podľa hodnoty a následne podľa rastu v bode `point`. Zoznam funkcií však nemusí obsahovať len funkcie definované v danom bode. Najprv je potrebné skontrolovať, či sú definované v bode `point` a tie funkcie, ktoré nie sú sa vypíšu ako nedefinované. Metóda vypíše na štandardný výstup nasledovné tri riadky:

- `"undefined: <zoznam>"`, kde `<zoznam>` je zoznam nedefinovaných funkcií oddelených `", "` (čiarka a medzera), v poradí v akom sú v zozname `list`.
- `"sorted by value at point <point>: <zoznam>"`, kde `<point>` je `point` a `<zoznam>` je zoznam definovaných funkcií utriedených podľa hodnoty.
- `"sorted by rate of change at point <point>: <zoznam>"`, kde `<point>` je `point` a `<zoznam>` je zoznam definovaných funkcií utriedených podľa rastu.

Za posledným riadkom napíše znak nového riadka. Zoznam na vstupe metóda nesmie meniť.
Poznámka: V Jave matematicky nedefinované operácie s typom `double` vracajú hodnoty `Double.NaN`, `Double.POSITIVE_INFINITY` alebo `Double.NEGATIVE_INFINITY`.
Poznámka: Na vytvorenie komparátorov sa vám môže zísť nejaká metóda z triedy `Comparator`.

Odovzdávanie na testovač

Na testovač odovzdávajte ZIP archív obsahujúci priečinok `function` a v ňom všetky triedy balíka `function`. Všetky triedy by mali byť uložené v samostatnom súbore. Testovač bude kontrolovať správnosť vášho riešenia na vybraných vstupoch.

Príklady

V nasledujúcich príkladoch budeme využívať premennú `X` ako inštanciu triedy `Variable` a `df` ako inštanciu `DecimalFormat`:

```
Function X = new Variable();
DecimalFormat df = new DecimalFormat("#.#####");
```

Nech `f` reprezentuje funkciu $\left(1+\frac{4x}{2^0}\right)^2$:

```
Function f = new Power(new Plus(new Constant(1), new Division(
            new Multiplication(new Constant(4), X),
            new Power(new Constant(2), 0))),
            2);
```

Nasledujúci kus programu:

```
System.out.println(f);
System.out.println(df.format(f.evaluate(1)));
Function fs = f.simplify();
System.out.println(fs);
System.out.println(df.format(fs.evaluate(1)));
System.out.println(fs.derivative());
Function fsds = fs.derivative().simplify();
System.out.println(fsds);
System.out.println(df.format(fsds.evaluate(1)));
```

Vypíše

```
(1+((4*x)/(2^0)))^2
25
(1+(4*x))^2
25
(2*((1+(4*x))^1))*(0+((0*x)+(4*1)))
(2*(1+(4*x)))*4
40
```

Nech `g` reprezentuje funkciu $\ln 5^{\cos\frac{x}{2}}$:

```
Function g = new Logarithm(Math.E, new Exponential(5, new Cosine(
            new Division(X, new Constant(2)))));
```

Nasledujúci kus programu:

```
System.out.println(g);
System.out.println(g.simplify());
System.out.println(df.format(g.evaluate(Math.PI)));
System.out.println(g.derivative());
Function gds = g.derivative().simplify();
System.out.println(gds);
System.out.println(df.format(gds.evaluate(Math.PI)));
```

Vypíše (výpisy derivácií majú byť jeden riadok, boli rozdelené len tu v zadaní)

```
ln(5^cos(x/2))
ln(5^cos(x/2))
0
(1/((5^cos(x/2))*1))*(((5^cos(x/2))*1.60944)*((-sin(x/2))*(((1*2)-(x*0))/(2^2))))
(1/(5^cos(x/2)))*(((5^cos(x/2))*1.60944)*((-sin(x/2))*0.5))
-0.80472
```

Nasledujúci kus programu:

```
Function sine = new Sine(Variable.X);
Function sined4 = sine.derivative().derivative().derivative().derivative();
System.out.println(sined4.simplify());
```

Vypíše

```
sin(x)
```

Nasledujúci kus programu:

```
Function ZERO = new Constant(0);
Function h = new Minus(ZERO, new Minus(ZERO, new Minus(ZERO,
    new Minus(ZERO, Variable.X))));
System.out.println(h);
System.out.println(h.simplify());
```

Vypíše

```
0-(0-(0-(0-x)))
x
```

Nech `list` reprezentuje zoznam funkcií $x, \frac{1}{x}, 2^x, \ln x, x^2+3, -3^{-x}$ vytvorený napríklad:

```
List<Function> list = List.of(X, new Division(new Constant(1), X),
    new Exponential(2, X), new Logarithm(2, X),
    new Plus(new Power(X, 2), new Constant(3)),
    new UnaryMinus(new Exponential(3, new UnaryMinus(X))));
```

Potom zavolanie `Function.sortFunctions(list, 0);` vypíše:

```
undefined: 1/x, log_2(x)
sorted by value at point 0: -(3^(-x)), x, 2^x, (x^2)+3
sorted by rate of change at point 0: (x^2)+3, 2^x, x, -(3^(-x))
```