Racionální kalkulačka

Ondřej Hruška

Semestrální práce
předmětu Programování 1

29. září 2013

Zadání

Program načte výraz, který může obsahovat celá čísla a operace sčítání, odčítání, násobení, dělení a mocnění a závorky, a provede jeho vyhodnocení s plnou přesností (výsledkem bude celé číslo nebo zlomek), s ohledem na prioritu operátorů a závorky; výraz je možné zadávat přímo programu nebo načítat / ukládat z / do souboru, stačí konzolové uživatelské rozhraní.

Rozbor

Vzhledem k složitosti úkolu jsem se rozhodl k problému přistoupit co nejobecněji, aby systém bylo možné snadno rozšiřovat a ladit. Projekt je rozdělen na dvě části – vyhodnocovací logiku a uživatelské rozhraní.

Vyhodnocení výrazu

Zadaný řetězec je nejprve zjednodušen a následně rozdělen na jednotlivé tokeny (čísla, operátory, závorky).

Z těchto tokenů je pak sestaven syntaktický strom, jehož vyhodnocením se získá výsledek zadaného vzorce.

Uživatelské rozhraní

Při tvorbě uživatelského rozhraní jsem se inspiroval linuxovým programem `bc`, což je jednoduchá terminálová kalkulačka.

Po spuštění se uživateli vypíše krátká informace o použití, další interakce probíhá formou zadávání příkazů.

Oproti zadání úlohy obsahuje aplikace některé nové prvky. Uživatel má například možnost vytvářet proměnné, měnit je a používat je uvnitř matematických výrazů. Dále stojí za zmínku rozšíření sady podporovaných operací o modulo (`%`) a faktoriál (`!`) a možnost vkládat desetinná čísla, která se převedou se na odpovídající zlomky (podobně jako celá čísla která jsou převedena na zlomky se jmenovatelem rovným jedné).

Po zadání příkazu „`help`” se vypíše stručná nápověda, která uživateli ozřejmí základní příkazy a podporované početní operace.

Základní příkazy

- „`decimal`” - změna výstupního formátu (zlomek / desetinné číslo)
- „`vars`” - vypíše seznam aktuálně uložených proměnných
- „`load filename`” - způsobí načtení a vykonání příkazů uložených v souboru daného jména.
- „`exit`” - ukončí aplikaci

Pro potřeby ladění a pro zajímavost jsou v aplikaci ponechány některé ladící příkazy:

- „`debug`” - aktivuje ladící režim (zobrazení seznamu tokenů a syntaktického stromu)
- „`utest`” - spustí jednotkové testy knihovny RCalc

Implementace

Zdrojový kód je bohatě zdokumentován formou *JavaDoc*, neměl by tedy být problém se v něm orientovat. Komentáře jsou v souladu s obecně doporučovanými postupy psány anglicky.

Rozdělení kódu

S cílem zajistit co největší modularitu a umožnit případné použití pro další projekty byl kód rozdělen na balíčky `rcalc` a `calculator`. Balíček `rcalc` poskytuje obecně použitelné třídy pro vyhodnocování výrazů a správu proměnných (slouží jako knihovna), zatímco balíček `calculator` obsahuje uživatelské rozhraní a hlavní třídu aplikace.

Průběh vyhodnocení výrazu

Po potvrzení zadaného řetězce klávesou *Return* je text převeden na malá písmena, jsou oříznuty bílé znaky na krajích a výraz je postupně porovnán s jednotlivými příkazy (např. `load`, `help`, `vars`, `exit` a pod.). Pokud se s některým shoduje, funkce přiřazená příkazu se vykoná.

V případě, že řetězec neodpovídá žádnému příkazu, je předán instanci třídy `RCalcSession`, která obstarává správu proměnných a zprostředkovává vyhodnocení výrazů třídou `Rcalc`.

> *Příklad zadaného výrazu:*
> `5(10+1)(4!*3^2)-12/7+2`

Třída `RCalc` se stará o vlastní vyhodnocení výrazu a chytání výjimek. Výraz je nejprve předán instanci třídy `Tokenizer`, která provede řadu úprav pomocí regulárních výrazů a následně řetězec rozdělí na tokeny. Tyto jsou vloženy do objektu typu `TokenList`, který rozšiřuje `ArrayList` z `java.utils`.

> *Zadaný výraz rozdělený na tokeny:*
> `5,*,(,10,+,1,),*,(,4,!,*,(,3,),^,2,),+,-12,/,7,+,2`

Zavoláním metody `parse()` na vzniklém `TokenListu` se zahájí tvorba syntaktického stromu.

Nejprve jsou ze seznamu tokenů vyjmuty obsahy závorek, které jsou následně vloženy do menších `TokenListů`, přeloženy a vráceny zpět. To je umožněno tím, že jak tokeny, tak i `TokenList` a všechny prvky syntaktického stromu implementují rozhraní `IToken`, je tedy možné je uchovávat ve společném seznamu.

Po zpracování závorek jsou postupně všechny tokeny představující aritmetické operátory nahrazeny instancemi tříd, které rozšiřují abstraktní třídu `Operation`. Každá operace si uchovává své operandy v soukromých proměnných, postupně tedy dochází k zjednodušení `TokenListu` na jedinou `Operation`, která tvoří kořenový uzel vzniklého syntaktického stromu.

Výsledný syntaktický strom:

```
ADD{
.    ADD{
.    .    MUL{
.    .    .    MUL{
.    .    .    .    5,
.    .    .    .    ADD{10,1}
.    .    .    },
.    .    .    MUL{
.    .    .    .    FCT{4},
.    .    .    .    POW{3,2}
.    .    .    }
.    .    },
.    .    DIV{-12,7}
.    },
.    2
}
```

Jak `Operation`, tak i `Fraction` (třída znázorňující zlomky) implementují rozhraní `IEvaluable`. Díky tomu stačí na kořeni syntaktického stromu zavolat metodu `evaluate()` a celý výraz se vyhodnotí. Návratovou hodnotou je výsledný zlomek.

Vypočtený výsledek:

```
83162/7
```

Datový typ „Fraction”

Pro znázornění zlomků slouží datový typ `Fraction` obsahující celočíselné proměnné představující čitatel a jmenovatel. Součástí zlomku jsou i metody pro různé aritmetické operace.

Všechna celá čísla jsou při parsování převedena na zlomky s čitatelem rovným jedné, čímž je vyhodnocování velmi zjednodušeno.

Vzhledem k požadavku na přesné vyjádření výsledku nebylo možné použít desetinné datové typy (`float`, `double`) a celočíselné typy (`int`, `long`) neposkytují dostatečný počet míst (např. pro výpočet vysokých mocnin či faktoriálu). Proto jsem pro uložení číselné hodnoty zvolil typ `BigInteger`, který dokáže pojmout i velmi vysoká celá čísla.

Pro případ, kdy uživatel potřebuje výsledek v desetinném formátu, má třída `Fraction` metodu `getDoubleValue()`, která vrátí přibližný výsledek typu `double`.

`Fraction` také poskytuje veřejné metody pro získání čitatele a jmenovatele jako `BigInteger`, což se může hodit při dalším programovém zpracování výsledku. Metoda `toString()` vrací řetězec ve formátu `čitatel/jmenovatel`.