Politechnika Warszawska
Wydział Elektryczny

Specyfikacja Implementacyjna „Arbitrage”

Autor:
Grzegorz Kopyt

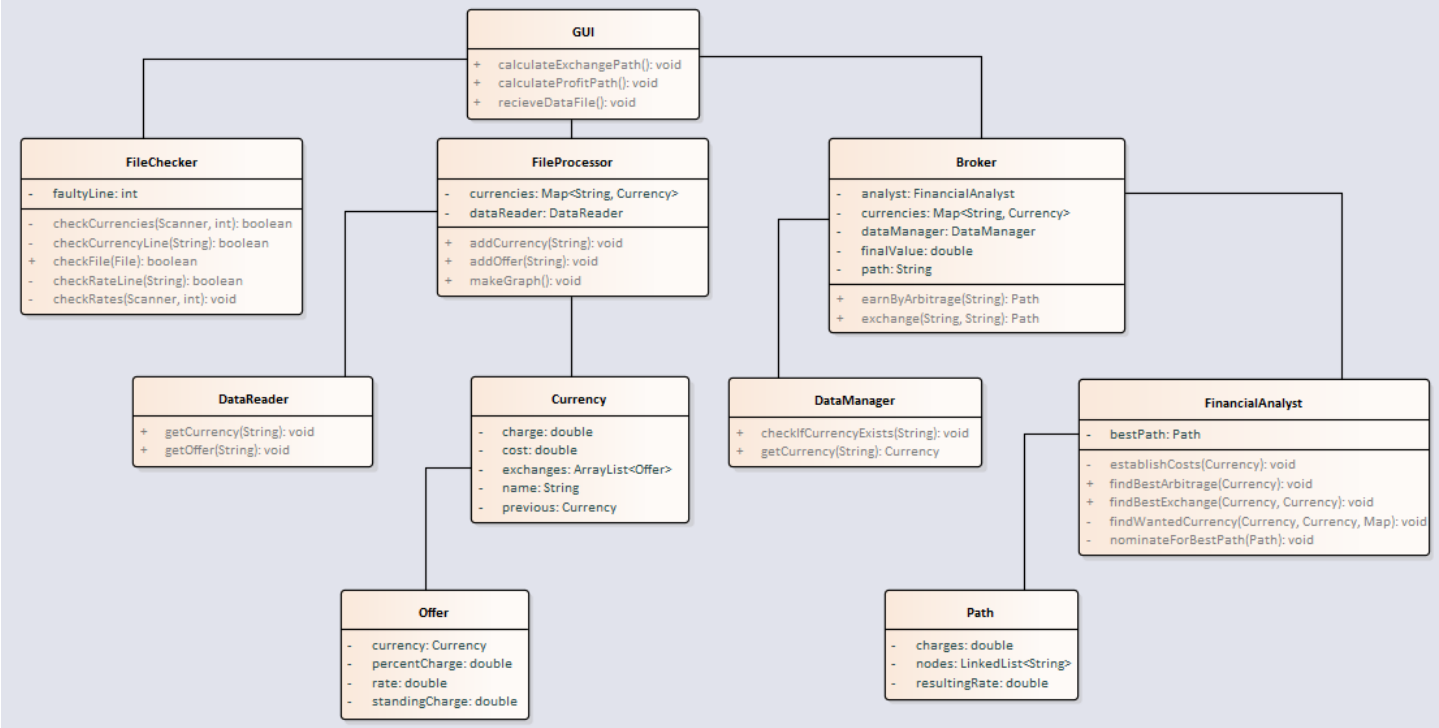
10 listopada 2018

Spis treści

1 Wstęp teoretyczny

Dokument ten dotyczy implementacji programu „Arbitrage". Został przygotowany w celu przedstawienia pomysłu na algorytm realizujący znajdowanie korzystnej ścieżki wymiany walut oraz dowolnego arbitrażu. Ponadto dokument informuje o technologiach, w których program będzie zrealizowany, testach jakie powinny zostać przeprowadzone oraz sprzęcie, na którym zostanie wykonany i uruchomiony.

2 Diagram klas



3 Opis algorytmu

3.1 Wczytanie danych

Dane wczytywane będą z pliku tekstowego, wykonanego według wzoru podanym w specyfikacji funkcjonalnej. Plik ten będzie zawierał definicje walut oraz kursy ich wymiany. Na początku zostanie sprawdzony pod kątem zgodności ze wzorem. Jeśli plik będzie wadliwy, algorytm przerwie swoje działanie, a w przeciwnym wypadku będzie kontynuował prace.

Algorytm umieści wszystkie zdefiniowane skrócone nazwy walut w *HashMapie* jako obiekty *Currency*. Dodatkowo każdy obiekt *Currency*, będzie zawierał *Liste* obiektów klasy *Offer*, w których zawarte będą informacje o kosztach i kursie wymiany na inną walutę.

Przykład:

```
EUR USD 1,13 STAŁA 1
USD PLN 0,8 PROC 0.025
PLN EUR 0,25 STAŁA 0.2
```

Obiekt *Currency* reprezentujący walutę *EUR*, będzie zawierał *Liste*, w której będzie obiekt *Offer*.
Obiekt *Offer* będzie zawierał:

- referencję do obiektu *Currency* reprezentującego *USD*,
- informację o kursie wymiany *EUR* na *USD* równym 1.13,
- informacje o opłacie stałej równej 1,
- informacje o opłacie procentowej równej 0 (bo jej nie ma w tym przypadku).

Dla innego zestawu danych na tej *Liście* może pojawić się więcej obiektów *Offer* zawierających informacje o wymianie *EUR* na inne waluty.

W ten sposób obiekty *Currency* i ich *Listy* obiektów *Offer* utworzą graf albo wiele grafów w zależności od danych wejściowych i możliwości wymian między walutami.

3.2 Koszt ścieżki

Kluczowym zadaniem algorytmu, będzie znajdowanie najkorzystniejszych ścieżek po grafie walut. W celu określenia


```
EUR USD 1,13 STAŁA 1
USD PLN 0,8 PROC 0.025
PLN EUR 0,25 STAŁA 0.2
```


Koszt dotarcia do danego węzła od wybranego węzła początkowego, będzie przechowywany Głównym celem jest, aby po skończonej pracy algorytmu w obiektach *Currency* pola *cost* i *charge* miały takie wartości, które kwotę końcową w walucie docelowej pozwolą obliczyć według wzoru:
k - kwota początkowa w początkowej walucie
w - kwota końcowa w walucie docelowej
w = k/cost - charge

Wartości pól *cost*, *charge* oraz *previous* w obiektach *Currency* sa zainicjowane wartościami *null*.

Algorytm operuje na grafie, w którym węzłami są obiekty *Currency*, a gałęziami obiekty *Offer*, a jego działanie jest następujące:

1. Obiekt klasy *Broker* otrzymuje od *GUI* informacje jakie zadanie ma zrealizować i na jakich walutach operować. Są dwa warianty pracy: *wymiana waluty*(patrz 2a.) lub *arbitrage*(patrz 2b.).
2. (a) Wymiana waluty
    i. *Broker* zleca *FinancialAnalyst*, aby ustawił koszty odwiedzenia węzłów w grafie zaczynając od podanej przez użytkownika waluty wyjściowej.

A. Z HashMapy *currencies* pobiera Set kluczy, z którego robi tablicę, kluczy.

B. Zaczynając od węzła początkowego iteruje po tablicy w pętli (jeśli napotka koniec tablicy zaczyna od początku).

C. Ustawia *cost* i *charge* węzła początkowego na 0.

D. Ustawia wartości pól *cost* i *charge* w węzłach sąsiadujących z węzłem początkowym według wzorów:

Cost:
newCost - wartośc pola *cost* w sąsiadującym węźle
cost - wartość pola *cost* w obecnym węźle
rate - wartość pola *rate* w gałęzi *Offer*
percent - wartość pola *percentCharge* w gałęzi *Offer*

newCost = cost/rate * (1-percent)

UWAGA! Jeśli wartość *cost* wynosi zero to:
newCost = rate * (1-percent)

Charge:
newCharge - wartośc pola *charge* w sąsiadującym węźle
charge - wartość pola *charge* w obecnym węźle
rate - wartość pola *rate* w gałęzi *Offer*
standingCharge - wartość pola *standingCharge* w gałęzi *Offer*

newCharge = charge/rate + standingCharge

Wartość *previous* sąsiadującego węzła ustawiamy na obecny węzeł.

UWAGA! Powyższych operacji dokonujemy, jeśli nowa wartość pola *cost* jest mniejsza od poprzedniej.

E. Następnie przechodzimy do kolejnego obiektu *Currency* w tablicy:
- Jeśli wartość jego pola *cost* wynosi *null* to przechodzimy do kolejnego obiektu w tablicy powtarzając podpunkt E.
- Jeśli wartość jego pola *cost* jest różna od *null* to powtarzamy czynności z podpunktów D, a potem E.

ii. Teraz aby podać najkrótszą drogę do waluty z waluty podanej wcześniej wystarczy wejść w walutę docelową i prześledzić szlak previous :) warto dodać na początek sprawdzenie na jaką walutę ustawiony jest graf, może nie będzie trzeba liczyć.

(b) Arbitrage

4 Diagram rozwiązania problemu

5 Opis ważniejszych metod

- boolean **establishCosts**(waluta początkowa)

Metoda ustawia w węzłach (Currencies) koszt dotarcia do tego węzła, wyruszając z węzła waluty początkowej. Działa według algorytmu Bellmana-Forda. Dodatkowo w każdym węźle zostawia informacje, która waluta jako ostatnia zaktualizowała koszt w danym węźle.

6 Testy

- **establishCosts**()
  - Metoda dostaje walutę, której nie ma w Mapie. Powinna zwrócić *false*.
  - Metoda dostaje *null*. Powinna zwrócić *false*.
  - Metoda dostaje walutę, która nie ma, żadnego połączenia z innymi. Powinna zwrócić true, a jej pola charge i cost powinny zawierać wartość zero.
  - Metoda dostaje walutę, która ma połączenia z trzema innymi walutami. Powinna zwrócić true i zmienić zawartości pól tych walut.

7 Informacje o sprzęcie i oprogramowaniu