Sprawozdanie końcowe
Projekt „Group of Visionaires”

Mateusz Smoliński

28 listopada 2018

Spis treści

1 Wstęp

Sprawozdanie powstało w celu podsumowania projektu „Group of Visionaires". Zawiera ono opis zastosowanego rozwiązania i algorytmu rozwiązującego problem arbitrażu i najkorzystniejszej wymianu. Opisane zostały także zmiany względem specyfikacji funkcjonalnej i implementacyjnej, które zostały podjęte już w trakcie implementacji projektu. Dokument zawiera także wnioski, jakie zostały wyciągnięte przy powstawaniu projektu.

Projekt „Group of Visionaires" ma za zadanie analizę sytuacji na giełdzie walutowej oraz wybór najkorzystniejszej ścieżki dla wymiany walut. Dodatkową funkcją jest wykrywanie ekonomicznego arbitrażu, czyli sytuacji, w której możliwa jest wymiana cykliczna waluty przynosząca zysk – możliwość włożenia pieniędzy w walucie X oraz wykonanie wymian, które zakończą się na tej samej walucie X z wartością pieniężną większą niż początkowa.

Program napisany w ramach tego projektu otrzymuje plik tekstowy. Powinien on zawierać wypisaną aktualną sytuację na giełdzie walut, według poniższego schematu:

```
# Waluty (id | symbol | pelna nazwa)
0 EUR euro
1 GBP funt brytyjski
2 USD dolar amerykanski
# Kursy walut (id | symbol (waluta wejsciowa) | symbol (waluta
wyjsciowa) | kurs | typ oplaty | oplata
0 EUR GBP 0.8889 PROC 0.0001
1 GBP USD 1.2795 PROC 0
2 EUR USD 1.1370 STALA 0.025
3 USD EUR 0.8795 STALA 0.01
```

Program powinien być uruchomiony z zestawem argumentów. Specyfikacja funkcjonalna zakładała uruchomienie go z 3 lub 4 argumentami – nazwa pliku z danymi, kwota, waluta wejściowa i ewentualnie waluta wyjściowa. Ze względu na zmianę w interpretacji polecenia program może być też uruchomiony z dwoma argumentami – w tym wypadku zostanie wyszukany dowolny arbitraż, jeżeli takowy występuje. Tym niemniej, funkcja opisana w specyfikacji funkcjonalnej (uruchomienie programu z podaniem jednej waluty i szukanie dla niej arbitrażu) wciąż może być używana przez użytkownika, co daje większą dowolność używania programu. Ostatecznie, program może otrzymać od 2 do 4 argumentów – nazwę pliku, kwotę wejściową i opcjonalnie 1 lub 2 waluty podane w postaci skrótów walutowych podanych w pliku.

Wynikiem działania programu jest dokładne wypisanie ścieżki wymiany walut – w zależności od liczby argumentów najkorzystniejszej wymiany walut lub korzystnego cyklu. Zostaje także wypisana końcowa wartość waluty po dokonaniu wymiany. W przypadku, gdy wymiana jest niemożliwa lub arbitraż nie został odnaleziony program wypisze stosowny komunikat. W przypadku wystąpienia błędu wynikającego ze źle wprowadzonych danych również zostanie wypisany komunikat, wskazujący na przyczynę wystąpienia błędu.

2 Działanie programu

2.1 Opis algorytmu

Algorytm zastosowany w finalnej wersji programu opiera się na przeszukiwaniu grafu wszerz (BFS). Traktując waluty jako wierzchołki grafu i kursy wymian jako krawędzie program przeszukuje graf w poszukiwaniu najkorzystniejszej wymiany.

Proces ten rozpoczyna się w wierzchołku reprezentującym walutę wejściową. W programie zamiast standardowej dla przeszukiwania BFS kolejki wierzchołków zastosowano kolejkę list – dróg, które potencjalnie mogą stanowić najkorzystniejsze ścieżki. Dopóki kolejka nie jest pusta (wszystkie ścieżki nie zostaną sprawdzone) kolejne ścieżki są wyjmowane z kolejki, po czym program sprawdza ich możliwe kontynuacje. Jeśli istnieją jeszcze nieodwiedzone w danej ścieżce wierzchołki i istnieje krawędź pozwalająca na przejście do nich, do kolejki trafiają nowe ścieżki stanowiące aktualnie rozpatrywaną ścieżkę oraz te nieodwiedzone wierzchołki. W przypadku odnalezienia połączenia dokonywane jest obliczenie wyjściowej wartości waluty, po czym wynik jest porównywany z wymianą aktualnie uważaną za najkorzystniejszą. Jeśli ścieżka okaże się być lepsza, zastępuje tą zapisaną wcześniej.

Dla szukania arbitrażu algorytm przebiega bardzo podobnie, z kilkoma zastrzeżeniami:

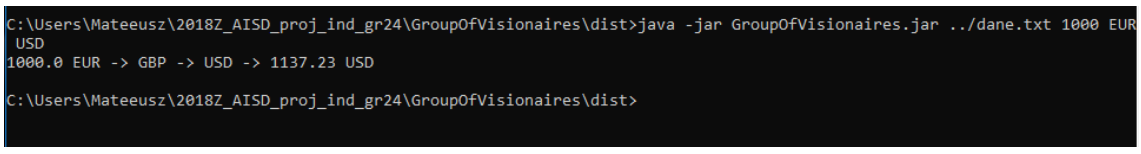
- do ścieżki może dołączyć wierzchołek odwiedzony, pod warunkiem, że jest to wierzchołek początkowy,
- za wierzchołek docelowy przyjmuje się wierzchołek początkowy,
- Po kalkulacji wyjściowej wartości waluty sprawdzane jest, czy jest ona większa od wartości początkowej.

Po zakończeniu działania algorytmu wynikiem jest wartość uznana za najwyższą. Jeśli wartość domyślna ścieżki nie została nadpisana, program dostanie informację o braku połączenia lub arbitrażu.

2.2 Przykład działania

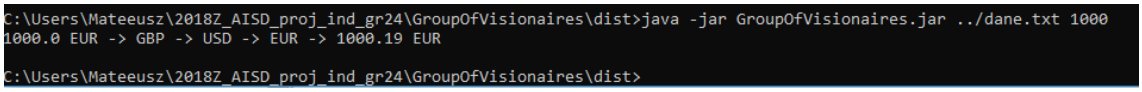
Poniższe przykłady zostały wykonane na komputerze opisanym w specyfikacji implementacyjnej, uruchomione w cmd.exe.

Pierwszy rysunek ukazuje poprawne działanie programu dla pliku z danymi dane.txt zawierającego dokładnie te same dane, które zostały umieszczone we wstępie tego sprawozdania.



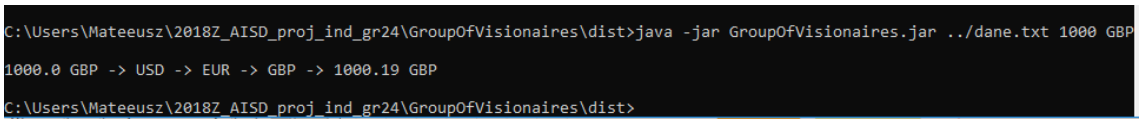
Rysunek 1: Uruchomienie programu z wyszukiwaniem ścieżki

Kolejny obraz przedstawia uruchomienie programu dla tych samych danych, ale z poszukiwaniem dowolnego arbitrażu.



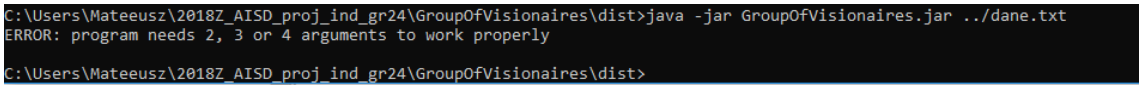
Rysunek 2: Uruchomienie programu – dowolny arbitraż

Na trzecim rysunku widzimy uruchomienie programu w trzeciej opcji – szukania arbitrażu dla konkretnej waluty.



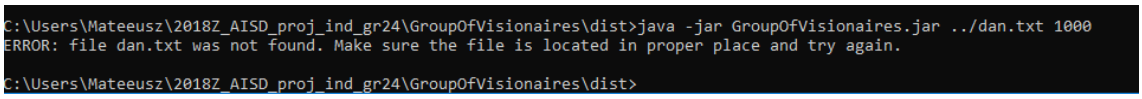
Rysunek 3: Uruchomienie programu – konkretny arbitraż

Kolejny obrazek to już niepoprawne uruchomienie programu – w tym wypadku użytkownik podał za mało danych, zabrakło przynajmniej podania kwoty wejściowej.



Rysunek 4: Uruchomienie programu – za mało danych

Piąty rysunek to próba uruchomienia programu dla pliku, który nie istnieje.

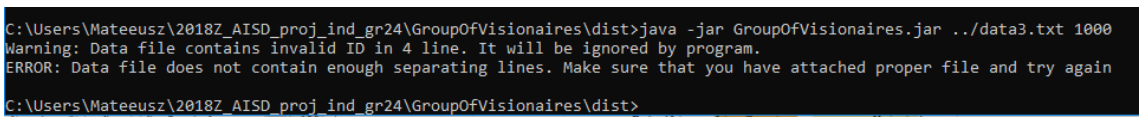


Rysunek 5: Uruchomienie programu – niepoprawne dane

Dwa kolejne obrazki to przykłady uruchomienia programu dla plików, które nie zostały poprawnie zapisane. Są to pliki użyte do testowania programu, których treść zamieszczono poniżej. Obrazki ukazują reakcję programu na konkretne błędy w pliku tekstowym.

Pierwszy plik to data3.txt:

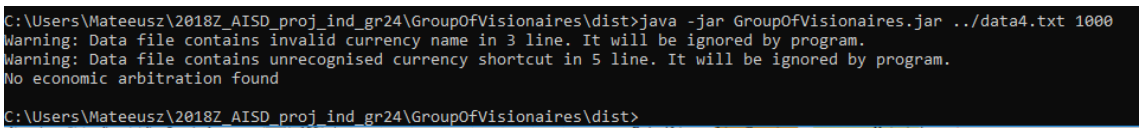
```
# Waluty (id | symbol | pelna nazwa)
0 EUR Euro
1 PLN Zloty
 Kursy walut (id | symbol (waluta wejsciowa)...
0 EUR PLN 0.8889 PROC 0.0001
```



Rysunek 6: Działanie programu z błędem krytycznym w pliku

Kolejny plik to data4.txt:

```
# Waluty (id | symbol | pelna nazwa)
0 EUR Euro
1 PLN
# Kursy walut (id | symbol (waluta wejsciowa)...
0 EUR PLN 0.8889 PROC 0.0001
```



Rysunek 7: Działanie programu z mniej ważnym błędem w pliku

3 Zmiany względem specyfikacji funkcjonalnej

3.1 Argumenty wywołania

Tak, jak zostało już wymienione w poprzedniej części sprawozdania, program może być teraz wywołany z tylko dwoma argumentami – w takim wypadku wyszukiwany jest dowolny arbitraż. W przypadku wywołania programu w taki sposób program wypisze jeden arbitraż dla pierwszej waluty, dla jakiej występuje – na przykład, jeśli w pliku z danymi znajduje się 6 walut i arbitraże występują dla waluty z numerami 2 i 4, program wypisze arbitraż dla waluty z ID równym 2.

3.2 Sytuacje wyjątkowe

W pierwotnej koncepcji w przypadku niewłaściwych danych w pojedynczej linii pliku z danymi program miał wypisać komunikat: „Warning: Data file contains invalid data in <number_of_line> line. It will be ignored by program.” Ze względu na zastosowanie wyjątków, które jest opisane szerzej w zmianach względem specyfikacji implementacyjnej, komunikaty wypisywane przez program są dokładniejsze. Na przykład, gdy w pliku z danymi wartość opłaty zawiera niewłaściwy znak, na przykład literę, program wypisuje komunikat: „Warning: Data file contains invalid fee value in <number_of_line> line. It will be ignored by program.” Może to ułatwić odnalezienie i poprawienie błędu użytkownikowi.

Specyfikacja funkcjonalna nie uwzględniała też przypadków niepożądanych, takich jak:

- opłata procentowa przekraczająca 1,
- kurs z waluty X na tą samą walutę X,
- kurs napisany dwukrotnie.

Ostateczna wersja programu ignoruje takie linie, również wypisując odpowiadające problemowi komunikaty.

4 Zmiany względem specyfikacji implementacyjnej

W strukturze całego programu nastąpiła jedna istotna zmiana – ze względu na wielkość pakietu *groupofvisionaires* przechowującego pierwotnie wszystkie klasy programu został wydzielony drugi pakiet, nazwany *filereader*, przechowujący klasę *DataReader* oraz wyjątki *ReadDataException* oraz *ParseLineException* używane przez program przy obsłudze błędów związanych właśnie z odczytem pliku. Ich dokładniejsze zastosowanie opisano poniżej, w sekcjach opisujących metody klasy *DataReader*.

4.1 Klasa CurrencyMatrix

Klasa odpowiedzialna za przechowywanie danych wczytanych z pliku nie uległa dużej zmianie względem specyfikacji implementacyjnej. Otrzymała jedynie jedno dodatkowe pole w postaci podawanej na początku istnienia klasy liczby n, odpowiadającej za ilość walut. Do metod dostępowych doszła zatem metoda *getN*, która wypisuje tą liczbę w innym miejscu kodu – w szczególności w trakcie iterowania po tablicy walut *currencies*, która jest prywatna i nie można dostać jej długości z zewnątrz.

4.2 Klasa DataReader

Klasa wczytująca dane z pliku została rozbudowana i ostatecznie składa się ona z trzech metod.

4.2.1 Metoda readData

- `public static CurrencyMatrix readData (File f) throws ReadDataException`

Metoda *readData* zgodnie ze wcześniejszym założeniem zwraca przygotowany obiekt klasy *CurrencyMatrix* w przypadku powodzenia. Jeżeli w trakcie wczytywania wystąpi problem uniemożliwiający poprawne działanie programu, metoda wyrzuca wyjątek *ReadDataException*, który jest odczytywany w metodzie *main* klasy *GroupOfVisionaires*.

4.2.2 Metoda parseCurrency

- `private static Currency parseCurrency(String line, int lineCounter) throws ParseLineException`

Metoda *parseCurrency* powstała na późniejszym etapie projektu, w celu poprawy czytelności i usprawnienia obsługi błędów w klasie *DataReader*. Otrzymuje ona wczytaną linię z pliku z danymi oraz numer linii, która jest aktualnie wczytywana. W przypadku poprawnej analizy linii metoda zwraca obiekt klasy *Currency*, gotowy do użycia w dalszej części programu.

Jeśli w trakcie wczytywania nastąpi problem, metoda wyrzuca wyjątek *ParseLineException*, który jest odbierany przez metodę *readData*. W przeciwieństwie do wyjątku *ReadDataException* nie powoduje on zakończenia próby wczytania pliku, a jedynie zignorowanie błędnej linii i przejście do kolejnego etapu programu.

4.2.3 Metoda parseRate

- `private static void parseRate(String line, int lineCounter, CurrencyMatrix currencyMatrix) throws ParseLineException`

Metoda *parseRate* powstała na późniejszym etapie projektu, podobnie jak poprzednia metoda, w celu oddzielenia obszernej procedury analizy linii z wymianą dwóch walut. W przeciwieństwie do metody *parseCurrency* nie zwraca ona przygotowanego obiektu, gdyż wybór indeksów w macierzy *CurrencyMatrix* jest bezpośrednio powiązane z odczytem tej linii oraz obsługą błędów. Otrzymuje ona wczytaną linię z pliku z danymi, numer linii, która jest aktualnie wczytywana oraz macierz *CurrencyMatrix*.

Jeśli w trakcie wczytywania nastąpi problem, metoda wyrzuca wyjątek *ParseLineException*, który jest odbierany przez metodę *readData*. Podobnie jak w przypadku metody *parseCurrency* nie powoduje on zakończenia próby wczytania pliku, a jedynie zignorowanie błędnej linii i przejście do kolejnego etapu programu.

4.3 Klasa ExchangeSearcher

- `public String SearchForArbitrage(double amount, CurrencyMatrix cm, Currency currency)`

- `public String SearchForBestExchange(double amount, CurrencyMatrix cm, Currency inputCurrency, Currency outputCurrency)`

Metody znajdujące się w tej klasie nie uległy dużej zmianie. Wykonują one dokładnie takie same czynności, jakie były przewidziane na wcześniejszym etapie projektu. Jedyną różnicą jest to, że metody nie wypisują na standardowym wyjściu odnalezionych ścieżek (lub komunikatów), ale zwracają je do metody *main* w postaci obiektu klasy String.

4.4 Klasa GroupOfVisionaires

Również klasa główna projektu przeszła pewne zmiany w stosunku do planów ze specyfikacji implementacyjnej. Podstawową różnicą jest to, że zyskała ona pola odpowiadające danym odbieranym od użytkownika – zarówno tych z argumentów wywołania, jak i odczytywanych z pliku, co ułatwiło przekazywanie tych danych w programie.

4.4.1 Metoda checkArgs

- `public boolean checkArgs(String[] args)`

Metoda sprawdzająca argumenty wywołania została przygotowana tak, żeby realizować wykonanie programu także dla dwóch argumentów – dla wyszukiwania dowolnego arbitrażu. Ze względu na testy jednostkowe w programie w finalnej wersji programu metoda ta jest dostępna publicznie.

4.4.2 Metoda main

- `public static void main(String[] args)`

Metoda *main* nie zmieniła się znacząco, poza obsługą wyszukiwania dowolnego arbitrażu dla dwóch podanych przez użytkownika argumentów. Występuje w niej obiekt klasy *GroupOfVisionaires*, który funkcjonuje jak kontener dla danych i to na nim uruchamiana jest metoda *checkArgs*.

5 Wnioski

1. Enumeracja w programie wpłynęła pozytywnie na czytelność kodu. Zastosowanie jej do definiowania typu opłat, jak i stanu metody wczytującej pozwoliła na sprawne i czytelne sterowanie programem.

2. Implementacja listy list w metodach wyszukujących ścieżek mogła nie być najwydajniejszym rozwiązaniem, jednak program działa bardzo szybko nawet dla wielu danych.

3. Zastosowany system wyjątków przy odczytywaniu pliku okazał się być bardzo wygodny. Poprawia on czytelność kodu i ułatwia przepływ sterowania w programie, w szczególności, gdy metoda ma zwracać inną wartość.

4. Samo opracowanie algorytmu wyszukującego ścieżkę w programie nie stanowiło dużego problemu. Znacznie więcej czasu pochłonęła dokładna obsługa błędów oraz testowanie programu pod kątem różnych możliwych scenariuszy.

5. Testy jednostkowe ukazały wiele błędów popełnionych w kodzie, ale testy akceptacyjne dokonane poza IDE ukazały właściwe działanie programu, pozwalając na poprawę rzeczy, na które wcześniej nie zwracało się uwagi.