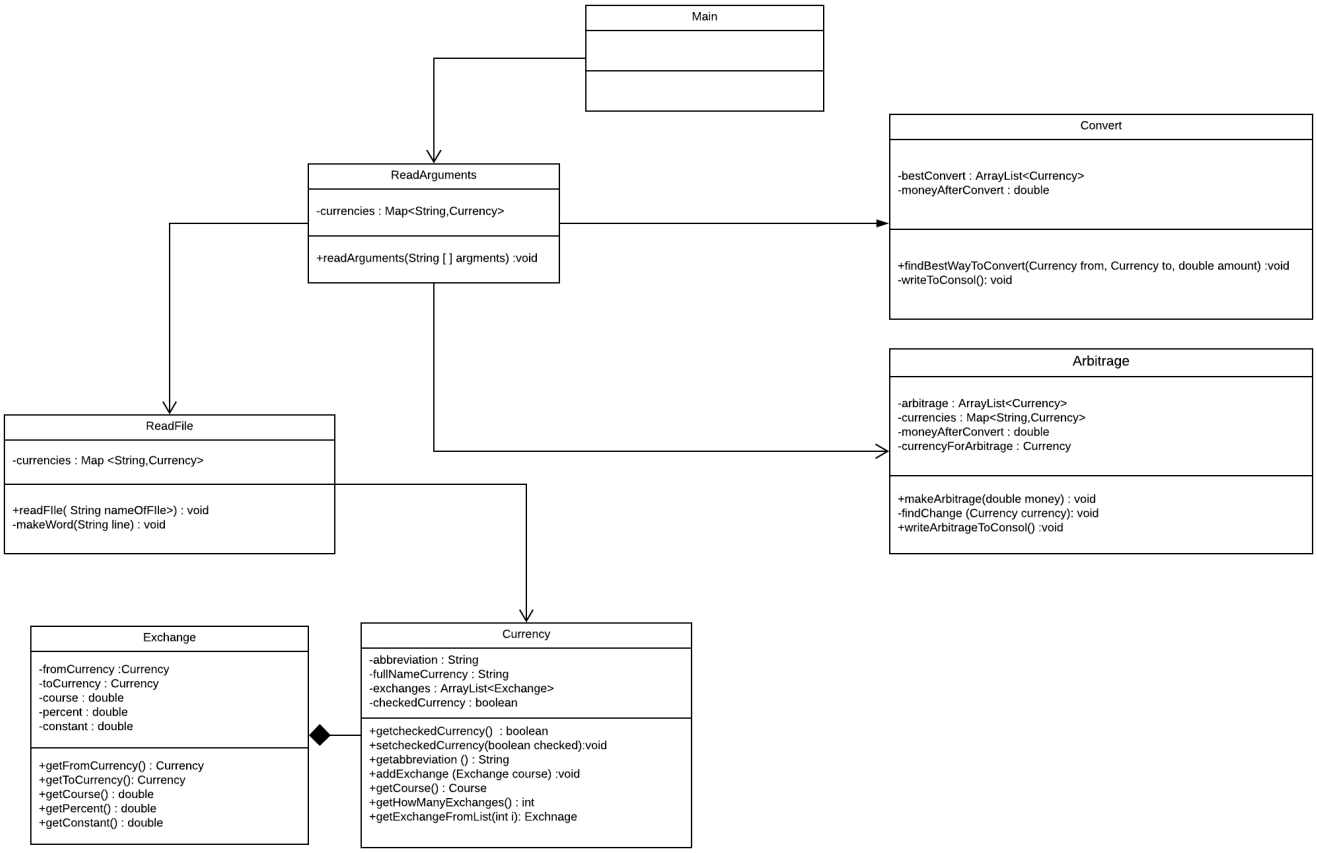
Politechnika Warszawska
Wydział Elektryczny
Kierunek Informatyka

Specyfikacja Implementacyjna

Wykonał: Aliaksandr Karolik (295138)
Warszawa, 10.11.2018

Contents

1 Diagram Klas



RYS.01 DIAGRAM KLAS

2 Opis metod i pól klas

2.1 Klasa Currency

Klasa reprezentująca walutę. Klasa będzie składać się z następujących pól:

```
private String abbreviation;
private String fullNameCurrency;
```

Pola przedstawiające nazwy waluty skruconą oraz pełną.

```
private ArrayList<Exchange> courses;
```

ArrayList przechowujący wszystkie możliwe wymiany dla danej waluty.

```
private boolean checkedCurrency;
```

Pole o typie boolean przechowujące wiedzę czy była zrobiona wymiana do tej waluty.

Będzie zawierać metody:

```
public addExchange (Exchange exchange);
```

Metoda dodająca do listy kurs wymiany danej waluty do jakieś innej. Jako argument metoda dostaje obiekt typu *Exchange*.

Ostatnie metody to metody typu set i get. Wszystkie te metody służą do ustawienia oraz pobierania wartości poszczególnych pól obiekta.

2.2 Klasa Exchange

Klasa reprezentująca kurs wymiany waluty. Klasa będzie składać się z następujących pól:

```
private Currency fromCurrency;
private Currency toCurrency;
```

Pola przechowujące informację z jakiej waluty odbywa się wymiana oraz do jakiej waluty jest ta wymiana.

```
private double course;
private double percent;
private double constant;
```

Pola przechowujące dane wymiany czyli: kurs wymiany, oprocentowanie przy wymianie lub stała która jest pobierana przy wymianie waluty. Klasa składa się z metod dostępniczych do pól.

2.3 Klasa ReadArguments

Klasa preprwodzająca analizę argumentów podanych przez użytkownika.
Klasa będzie mieć jedno pole:

```
private Map<String,Currency> currencies;
```

Pole o typie *Map* będzie przechowywać waluty. Jako klucz będzie stasowana nazwa skrucona waluty oraz elementami mapy będzą obiekty klasy *Currency*.
Klasa będzie mieć metodę:

```
public void readArguments(String [] arguments);
```

Metoda analizująca podane argumenty przez użytkownika. Po przeprowadzeniu analizy argumentów metoda deceduje w jakim trybie muszi działać program i wywoła odpowiednie metody.

2.4 Klasa ReadFile

Klasa która przeczyta podane na wejście przez użytkownika plik zawierający waluty oraz kursy wymiany tych walut. Klasa będzie miała następujące pole:

```
private Map <String,Currency> currencies;
```

Pole do którego klasa będzie zapisywać waluty zczytane z pliku.
Klasa *ReadFile* będzie składa się z następujących metod:

```
public void readFile(String nameOfFile);
```

Metoda która preprowadzi analizę pliku wejściowego podanego przez użytkownika. Metoda jako argumenty otrzymuje nazwę pliku wejściowetgo.

```
private void makeWords(String line);
```

Wewnętrzna pomocnicza metoda używana dla analizy pobranych linii z pliku wejściowego.

2.5 Convert

Klasa będzie składać się z następujących pól:

```
private ArrayList<Currency> bestConvert;
private double moneyAfterConvert;
```

Lista o nazwie *bestConvert* będzie przechowywać ciąg walut które będzą tworzyć najkorzystnejszą ścieżkę wymiany waluty. Pole o nazwie *moneyAfterConvert* będzie przechowywać kwotę która zostanie otrzymana po wykonaniu wszystkich wymian.
Klasa będzie mieć następujące metody:

```
public void findBestWayToConvert(Currency from,Currency to,
                                                    double amount);
```

Metoda będzie szukać najkorzystniejszą ścieżkę wymiany waluty. Dokładnie jak ona będzie szukać ścieżkę zostanie opisane w rozdziale *Opis działania algorytmu*. Jako argumenty metoda *findBestWayToConvert* dostaje *Currency from* walutę z której użytkownik chce przekonwertować pieniądze, *Currency to* walutę do której użytkownik chce przekonwertować pieniądze oraz *double amoutn* kwotę którą użytkownik chce przekonwertować.

```
public void writeToConsol();
```

Metoda wyświtlająca w konsoli najkorzystniejszą ścieżkę.

2.6 Klasa Arbitrage

Klasa będzie składać się z następujących pól:

```
private ArrayList<Currency> arbitrage;
private Map<String,Currency> currencies;
private double moneyAfterConvert;
private Currency currencyForArbitrage;
```

Lista o nazwie *arbitrage* będie przechowywać walute tworzące arbitraż. Zmienna o nazwie *currencies* typu *Map* będzie zawierać w sobie wszystkie walute podane przez użytkownika. Pole o nazwie *moneyAfterConvert* będzie przechowywać kwotę która wyjdzie po arbitrażu. Pole o nazwie *currencyForArbitrage* będzie przechowywać kwotę podaną przed użytkownika.

Klasa będzie składać się z następujacyhc metod:

```
public void makeArbitrage (double money);
```

Metoda będzie szukać dowolny arbitraż dla walut zawartych w *currencies*. Dokładnie jak ona będzie szukać arbitraż zostanie opisane w rozdziale *Opis działania algorytmu*. Metoda jako argument dostaję kwote dla której będzie szukany arbitraż.

```
private void findChange(Currency currency);
```

Wewnętrzna metoda używana dla znalezienia możliwej wymiany.

```
public void writeArbitrageToConsol();
```

Metoda wyświetlająca w konsoli arbitraż zawarty w liście o nazwie *arbitrage*.

3 Opis działania algorytmu

Algorytm zacznie swoje działanie z przeczytania podanego na wejście pliku przez użytkownika. Jeżeli plik będzie zawierać błędy program wyświetli komunikat w konsoli i przerwie swoje działanie. Algorytm czytając plik będzie dodawać do Mapy obiekty typu *Currency*. Każdy obiekt typu *Currency* będzie zawierał w sobie *ArrayList* obiektów klasy *Exchange*, w których zawarte będą informacje o kosztach i kursie wymiany na inną walutę.

3.1 Znalezienie najkorzystniejszej ścieżki wymiany waluty

Algorytm będzie zrealizowany w sposób rekurencyjny. Początkowa waluta oraz początkowa kwota będą zapisane do zmiennych pomocnyczych. Następnie waluta która podana do metody jako waluta z której szukamy ścieżke będzie zapisana do wewnętrznej listy do której będą zapisywane waluty tworzące ścieżkę wymian.

Następnie program będzie przechodzić iteracyjnie po liście możliwych wymian którą ma każda waluta. Pogram będzie robić wymianę do waluty z listy jeżeli waluta nie będzie oznaczona, że już uczestniczy w ścieżce. Następnie program będzie znów wywoływać metodę wyszukiwania najkorzystniejszej ścieżki dla aktualnej waluty. Dzięki tej rekurencji zostaną przeprowadzone wszystkie możliwe ścieżki wymian.

Program wyjdzie z pętli po elementam listy możliwych wymian wtedy gdy wszystkie waluty już uczestniczą w ścieżce. Po wyjściu z pętli program sprawdzi czy udało się przejść do waluty końcowej do której użytkownik chciał przekonwertować pięniądze. Jeżeli końcowa waluta będzie poprawna program zapisze wyjściową kwotę oraz ścieżke.

Jeżeli już była znaleziona ścieżka doprowadzająca do waluty wyjściowej to nowa otrzymana kwota będzie porównana z aktualną maksymalną kwotą. Jezeli nowa otrzymana kwota będzie większa od aktualnej to zostanie stwierdzono że znaleziona nowa najkorzystniejsza ścieżka.

3.2 Znalezienie dowolnego arbitrażu

Algorytm będzie zrealizowany w sposób rekurencyjny. Metoda realizująca algorytm dostanie jako argumenty: kwotą podaną przez użytkownika oraz obiekt typu *Map* zawierający wszystkie waluty.

Początkowa kwota będie zapisana do wewnętrznej zmiennej pomocniczej. Program będzie przechodzić iteracyjnie po mapie zawierającej wszystkie waluty. Dla każdej początkowej waluty program będzie przechodzić po liście możliwych wymian którą ma każda waluta. Pogram będzie przeprowadzać wymianę do waluty z listy jeżeli waluta jeszcze nie uczestniczy w ścieżce wymian.

Po wymianie program będzie szukał dla otrzymanej waluty możliwe przejście do waluty początkowej z której zaczynał się arbitraż. Jeżeli uda się

znaleźć takie przejście program wykona wymianę, a później porówna końcową kwotę z początkową. Jeżeli kwota końcowa będzie wększa to zostanie stwierdzono że arbitraż został znaleziony.

W przeciwnym przypadku program będzie znów wywoływać metodę która będzie szukać możliwą wymianę dla aktualnej waluty. Dzięki tej rekurencji zostaną przeprowadzone wszystkie możliwe ścieżki wymian dla początkowej waluty wziętej z mapy.

4 Testy jednostkowe

Testowane będą główne klasy programu:

1. Klasa Arbitrage,
2. Klasa Convert,
3. Klasa ReadFile.

4.1 Testowanie metody findBestWayToConvert z klasy Arbitrage

Testowanie metody dla wyszukiwania dowolnego arbitrażu będzie przeprowadzono dla następnych scenariuszew:

1. Testowanie na poprawnych danych. Testowany przypadek gdy dane są poprawne i dobrane tak, że będą zawierać tylko jeden możliwy arbitraż. Test zostanie zaliczony gdy metoda znajdzie ten arbitraż. W innych przypadkach test zostanie nie zaliczony.
2. Testowanie na danych w których nie ma arbitrażu. Testowany przypadek gdy dane dobrane tak że nie zawierają możlich przejść dla arbitrażu. Test zostanie zaakceptowany gdy metoda wyświetli komunikat o tym że nie udało się znaleźć arbitraż.

4.2 Testowanie metody makeArbitrage z klasy Convert

Testowanie metody dla wyszukiwania najkorzystniejszej ścieżki wymiany waluty będzie przeprowadzono dla następnych scenariuszew:

1. Testowanie na poprawnych danych. Testowany przypadek gdy wszystkie dane są poprawne i dobrane tak, że będzie tylko jedna prawidłowa ścieżka. Test zostanie zaakceptowany gdy metoda znajdzie tą jedną ścieżkę.
   W innych przypadkach test zostanie nie zaliczony.
2. Testowanie na danych w których nie ma najkorzystniejszej ścieżki dla wymiany waluty. Testowany przypadek gdy dane dobrane tak, że nie istnieje najkorzystniejszej ścieżki wymiany waluty dla podanych argumentów. Test zostanie zaakceptowany gdy metod wyświetli komunikat o tym,

że nie istnieje najkorzystniejszej ścieżki wymiany waluty dla danych wejściowych. W innych przypadkach test zostanie nie zaliczony.

3. Testowany przypadek gdy metod dostanie walutę której nie było w pliku wejściowym. Test zostanie zaakceptowany gdy metod wyświetli komunikat o tym, że użytkownik podał błędną walutę początkową. W innych przypadkach test zostanie nie zaliczony.

4.3 Testowanie metody readFile z klasy ReadFile

Testowanie metody dla przeczytywania pliku wejściowego będzie przeprowadzono dla następnych scenariuszew:

1. Plik zawiera błędne farmotowanie. Test zostanie zaakceptowany gdy metod wyświetli komunikat o błędzie w pliku wejściowym. W innych przypadkach test zostanie nie zaliczony.
2. Podanie nie istniejącego pliku. Test zostanie zaakceptowany gdy metod wyświetli komunika o tym że plik nie istnieje. W innych przypadkach test zostanie nie zaliczony.

5 Informacje o sprzęcie i oprogramowaniu

Program będzie pisany w języku Java w wersji Javy "10.0.1". Używane będzie zintegrowane środowisko programistyczne IntelliJ IDEA . Testy odbędą się na komputerze o następujących charakterystykach:

1. Procesor: Intel (R) Core (TM) i5-6200U CPU 2.30GHz 2.40 GHz,
2. Pamieci RAM: 8 GB,
3. Karta graficzna: NVIDIA GeForce 940M,
4. System operacyjny: Windows 10.