

Politechnika Warszawska  
Wydział Elektryczny

---

SPECYFIKACJA IMPLEMENTACYJNA  
„ARBITRAGE”

---

*Autor:*  
GRZEGORZ KOPYT

11 listopada 2018

---

## Spis treści

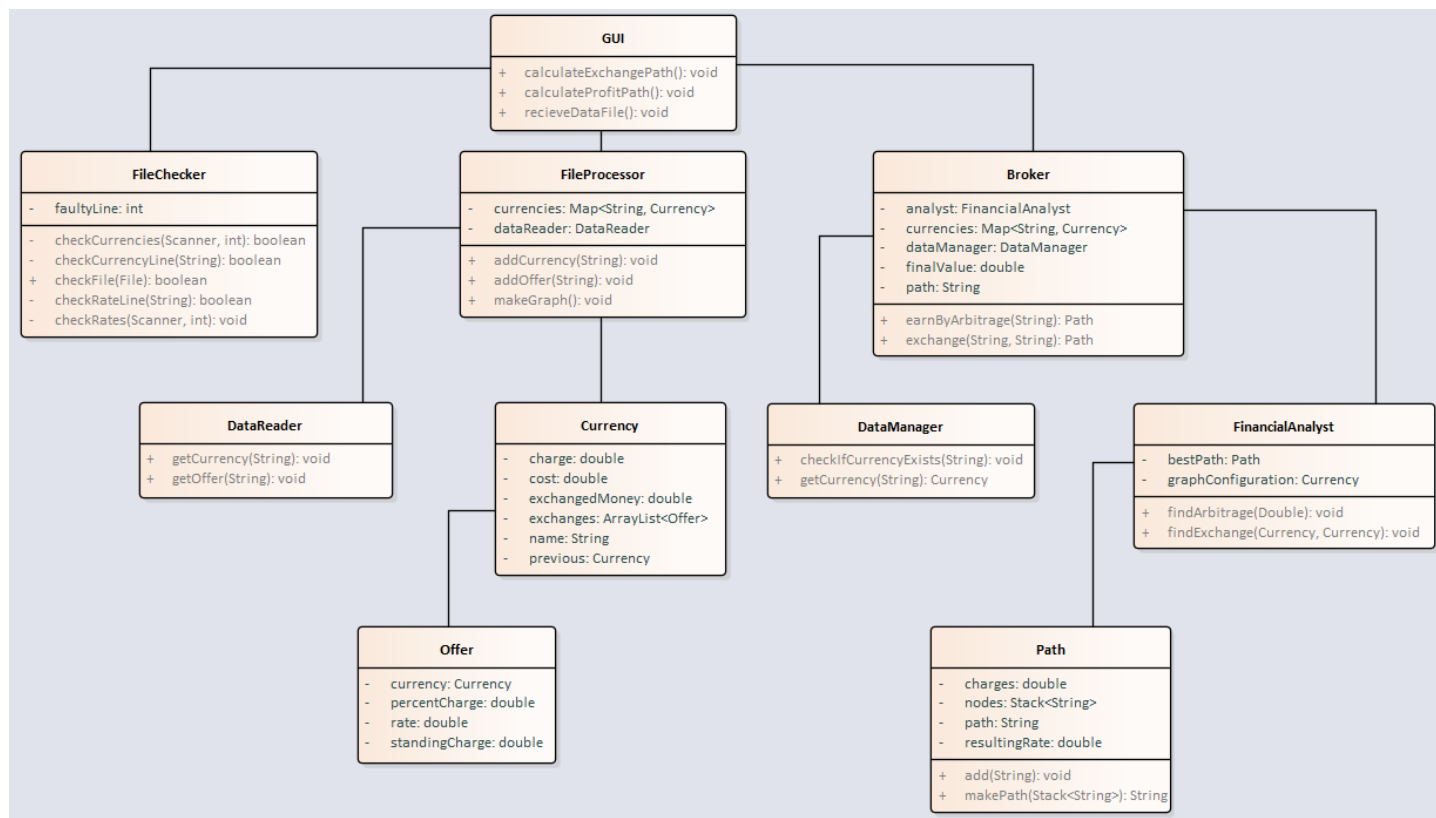
<b>1</b>	<b>Wstęp teoretyczny</b>	<b>1</b>
<b>2</b>	<b>Diagram klas</b>	<b>2</b>
<b>3</b>	<b>Opis algorytmu</b>	<b>2</b>
3.1	Wczytanie danych . . . . .	2
3.2	Koszt ścieżki . . . . .	3
3.3	Nadawanie węzłom kosztów . . . . .	4
3.3.1	Znajdowanie korzystnej ścieżki wymiany waluty . . . . .	4
3.3.2	Znajdowanie dowolnego arbitrażu . . . . .	5
3.3.3	Wyświetlanie ścieżki . . . . .	6
<b>4</b>	<b>Opis ważniejszych metod</b>	<b>6</b>
<b>5</b>	<b>Testy</b>	<b>7</b>
<b>6</b>	<b>Informacje o sprzęcie i oprogramowaniu</b>	<b>7</b>

---

## 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 *Listę* 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 *Listę*, 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.

Obiekty *Currency* będą węzłami, a obiekty *Offer* czymś na kształt gałęzi jako, że łączą ze sobą węzły.

### 3.2 Koszt ścieżki

Kluczowym zadaniem algorytmu, będzie znajdowanie najkorzystniejszych ścieżek po grafie walut. W wyborze najkorzystniejszej ścieżki pomoże nam koszt ścieżki. Koszt ścieżki będzie kosztem odwiedzenia danego węzła w zależności od węzła początkowego. W celu określenia kosztu odwiedzenia danego węzła, każdy węzeł będzie przechowywał informacje o koszcie dotarcia do niego.

Koszt ten będzie składał się z dwóch czynników:

- kosztu (*cost*) - "kurs"wymiany waluty początkowej na walutę obecnego węzła przez najkorzystniejszą ścieżkę (zawiera opłaty procentowe napotkane na ścieżce),
- opłaty stałe (*charge*) - w walucie obecnego węzła, wszystkie napotkane na ścieżce.

Przykład:

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

1. *Cost* dotarcia do *EUR* oraz jego *charge* będą równe zero, ponieważ od tego węzła zaczniemy.
2. *Cost* dotarcia do *USD* będzie równy 1.13, a jego *charge* będzie równa 1.
3. Następnie:
  - (a) *Cost* dotarcia do *PLN* będzie równy  $1.13 * 0.8 * (1-0.025) = 0.8814$ ,  
tzn. (*cost USD*) \* (kurs wymiany *USD* na *PLN*) \* (1 - opłata procentowa).
  - (b) *Charge* dotarcia do *PLN* będzie równa  $1 * 0.8 + 0 = 0.8$ ,  
tzn. (*charge USD*) \* (kurs wymiany *USD* na *PLN*) + opłata stała.
4. Następnie:
  - (a) *Cost* dotarcia do *EUR* będzie równy  $0.8814 * 0.25 * (1-0) = 0.22035$ ,  
tzn. (*cost PLN*) \* (kurs wymiany *PLN* na *EUR*) \* (1 - opłata procentowa).
  - (b) *Charge* dotarcia do *EUR* będzie równa  $0.8 * 0.25 + 0.2 = 0.4$ ,  
tzn. (*charge PLN*) \* (kurs wymiany *PLN* na *EUR*) + opłata stała.

Kiedy wszystkie węzły otrzymają swój najkorzystniejszy koszt ( o czym w dalszej części specyfikacji), wtedy wartości tych kosztów mogą posłużyć do obliczenia kwoty końcowej, jaką użytkownik uzyska z wymiany walut.

Przykład:

1. Węzłem początkowym było *EUR*.
2. 1000 *EUR* chcemy wymienić na *PLN*.
3. Sprawdzamy koszt *PLN*:
  - *cost* wynosi 0.8814,
  - *charge* wynosi 0.8 (w *PLN*, dlatego trzeba przeliczyć na *EUR*).

4. Z wymiany uzyskamy kwotę  $(1000 * 0.8814) - (0.8 / 0.8814) = 880.49$ ,  
tzn. kwota \* (*cost* PLN) - (*charge* / *cost* PLN).

Z powyższego przykładu wynika, że najkorzystniejsza ścieżka to taka, której *cost* jest jak największy, a *charge* jak najmniejszy. Łatwo zauważyć, że opłacalność ścieżki w dużej mierze zależy od kwoty, którą dysponujemy. Przykładowo, jeśli chcemy wymienić 1 *EUR* i otrzymamy 100 *PLN*, a opłata wynosi 100 *PLN*, to nie opłaca nam się zupełnie taka transakcja. Natomiast w przypadku, gdy wymieniamy 1000 *EUR* i otrzymamy 100 000 *PLN* to wtedy opłata 100 *PLN* nie robi nam różnicy, dlatego:

- w przypadku znajdowania najkorzystniejszej ścieżki wymiany waluty (nie znając kwoty wyjściowej), za najkorzystniejszą ścieżkę uznaję tę, której *cost* jest największy,
- w przypadku znajdowania dowolnego arbitrażu (znając kwotę wyjściową), algorytm uwzględni *charge* (o czym w dalszej części specyfikacji).

### 3.3 Nadawanie węzłom kosztów

Algorytm chodzenia po grafie i nadawania węzłom kosztów, będzie oparty o algorytm Bellmana-Forda. Na potrzeby śledzenia ścieżek wymian, obiekty *Currency* będą zawierać pole *previous* z referencją do waluty, która jako ostatnia zmodyfikowała koszt tego węzła (o czym w dalszej części specyfikacji). Nadawanie węzłom kosztów będzie różniło się w zależności od zadania realizowanego przez program (znajdowanie korzystnej ścieżki wymiany waluty lub znajdowanie dowolnego arbitrażu).

#### 3.3.1 Znajdowanie korzystnej ścieżki wymiany waluty

Koszty ścieżek dotarcia do danego węzła z węzła początkowego, są ściśle zależne od węzła początkowego. Z tego powodu w klasie *FinancialAnalyst* (odpowiedzialnej za szukanie korzystnej ścieżki wymiany waluty oraz dowolnego arbitrażu) znajdzie się zmienna *graphConfiguration*, która będzie informować o tym na jaki węzeł początkowy skonfigurowany (węzłom nadane są najkorzystniejsze koszty) jest obecnie graf. Pozwoli to w niektórych przypadkach na wykorzystanie pracy, którą program wykonał już wcześniej i nie będzie konieczności wykonywania poniższych instrukcji jeszcze raz.

W przeciwnym wypadku, jeśli graf nie jest jeszcze skonfigurowany albo skonfigurowano go na inny węzeł niż ten obecnie podany przez użytkownika jako waluta wyjściowa, algorytm wykonuje następujące operacje:

1. Algorytm bierze wszystkie waluty z *HashMapy* i wstawia je do *ArrayListy* (będzie po nich kolejno iterował w obiegu zamkniętym, czyli jak dotrze do końca *Listy*, następnym elementem jest zerowy, potem pierwszy itd.).

Na początku wartości *cost* i *charge* wszystkich węzłów są ustawiane na liczbę ujemną, a *previous* na wartość *null*.

2. Pierwszą walutą (węzeł początkowy), od której algorytm rozpocznie nadawanie kosztów, jest ta podana przez użytkownika jako waluta, którą użytkownik chce wymienić,

Wartości *cost* i *charge* węzła początkowego przyjmują wartość zero.

3. Obecny węzeł (obecna waluta) to waluta początkowa.
4. Flaga zmiany kosztu, która znajdować się będzie w metodzie realizującej to zadanie, zostaje wygaszona, jeśli obecny węzeł to węzeł początkowy.

Flaga zostaje zapalona w momencie kiedy wartość kosztu sąsiadującego węzła zostaje zmodyfikowana.

5. Następnie algorytm nadaje węzłom sąsiadującym z obecnym węzłem koszty w sposób opisany w podsekcji *Koszt ścieżki*, pod warunkiem, że nowy *cost* jest większy od bieżącego znajdującego się w sąsiadującym węźle. Jeśli ten warunek jest spełniony, w sąsiadującym węźle nadane zostają nowe wartości *cost* oraz *charge*, a także polu *previous* przypisana zostaje obecna waluta (ponieważ to ona spowodowała, że nowy *cost* jest większy od starego).

Węzeł początkowy może nadać wartości swoim węzłom sąsiadującym tylko raz. W kolejnych iteracjach pomijamy go, żeby nie wpaść w nieskończoną pętlę.

6. Po zaktualizowaniu kosztów w węzłach sąsiadujących z obecnym węzłem, obecnym węzłem staje się następna waluta według kolejności opisanej w punkcie pierwszym.
  - (a) Jeśli obecny węzeł jest węzłem początkowym, a flaga zmiany kosztu jest wygaszona, to znaczy, że w poprzedniej iteracji (jedna iteracja to rozpatrzenie każdego węzła z *ArrayListy* z punktu pierwszego) nie dokonano już żadnej zmiany, czyli nadano już najkorzystniejsze koszty wszystkim węzłom. Zatem korzystna ścieżka wymiany zostaje wyświetlona (o czym w dalszej części specyfikacji).
  - (b) Jeśli obecny węzeł jest węzłem początkowym, a flaga zmiany kosztu jest zapalona, to należy powtórzyć punkty 4-6, ponieważ jeszcze nie nadano najkorzystniejszych kosztów wszystkim węzłom.
  - (c) Jeśli w obecnym węźle wartości *cost* i *charge* są liczbami ujemnymi, a *previous* ma wartość *null*, to znaczy, że jeszcze nie wiemy jak dotrzeć do tego węzła i bez aktualizacji kosztów węzłów sąsiadujących z obecnym węzłem powtarzamy punkt 6.
  - (d) W pozostałym wypadku (różnym od a, b i c) powtarzamy punkty 5-6.

### 3.3.2 Znajdowanie dowolnego arbitrażu

W przypadku znajdowania dowolnego arbitrażu, przy wyszukiwaniu ścieżki takiego arbitrażu brana jest pod uwagę, także *charge* każdego węzła. W tym celu obiekty *Currency* będą zawierały pole *exchangedMoney*, które przedstawiać będzie kwotę w walucie danego węzła, jaką możemy uzyskać dokonując wymiany kwoty w walucie początkowej na walutę danego węzła po korzystnej ścieżce.

Wyszukiwanie arbitrażu działa następująco:

1. Algorytm bierze wszystkie waluty z *HashMapy* i wstawia je do *ArrayListy* (będzie po nich kolejno iterował w obiegu zamkniętym, czyli jak dotrze do końca *Listy*, następnym elementem jest zerowy, potem pierwszy itd.).
2. Na początku wartości *cost*, *charge* oraz *exchangedMoney* wszystkich węzłów są ustawiane na liczbę ujemną, a *previous* na wartość *null*.
3. Pierwszą walutą ( węzeł początkowy), od której algorytm rozpocznie nadawanie kosztów, jest ta, która na *Liście* z punktu pierwszego, ma indeks zerowy.

Wartości *cost* oraz *charge* węzła początkowego przyjmują wartość zero. Wartość *exchangedMoney* zostaje zainicjowana kwotą podaną przez użytkownika.

4. Obecny węzeł (obecna waluta) to waluta początkowa.
5. Flaga zmiany kosztu, która znajdować się będzie w metodzie realizującej to zadanie, zostaje wygaszona, jeśli obecny węzeł to węzeł początkowy.

Flaga zostaje zapalona w momencie kiedy wartość kosztu sąsiadującego węzła zostaje zmodyfikowana.

6. Następnie algorytm nadaje węzłom sąsiadującym z obecnym węzłem koszty w sposób opisany w podsekcji *Koszt ścieżki*, pod warunkiem, że kwota końcowa, (obliczona według wzorca z podsekcji *Koszt ścieżki*, na bazie nowych wartości *cost*, *charge* oraz wartości *exchangedMoney* z obecnego węzła) jest większa od bieżącej wartości *exchangedMoney* znajdującej się w sąsiadującym węźle.

Jeśli ten warunek jest spełniony, w sąsiadującym węźle nadane zostają nowe wartości *cost* oraz *charge*, a także polu *previous* przypisana zostaje obecna waluta ( ponieważ to ona spowodowała, że nowa wartość *exchangedMoney* jest większa od starej). Pole *exchangedMoney* otrzymuje wartość obliczonej kwoty końcowej.

Węzeł początkowy może nadać wartości swoim węzłom sąsiadującym tylko raz. W kolejnych iteracjach pomijamy go, żeby nie wpaść w nieskończoną pętlę.

7. Po zaktualizowaniu kosztów w węzłach sąsiadujących z obecnym węzłem, obecnym węzłem staje się następna waluta według kolejności opisanej w punkcie pierwszym.

- 
- (a) Jeśli obecny węzeł jest węzłem początkowym, a flaga zmiany kosztu jest wygaszona, to znaczy, że w poprzedniej iteracji (jedna iteracja to rozpatrzenie każdego węzła z *ArrayListy* z punktu pierwszego) nie dokonano już żadnej zmiany, czyli nadano już najkorzystniejsze koszty wszystkim węzłom.
- Jeżeli wartość *exchangedMoney* obecnego węzła (węzła początkowego) jest większa od wartości podanej przez użytkownika to znaleziono korzystny arbitraż i można wyświetlić jego ścieżkę (o czym w dalszej części specyfikacji) oraz kwotę.
  - Jeżeli wartość *exchangedMoney* obecnego węzła (węzła początkowego) nie jest większa od wartości podanej przez użytkownika to należy powtórzyć punkty 2-7 z tą różnicą, że w punkcie 3 wartością początkową staje się waluta o indeksie o 1 większym niż obecna (w *ArrayLiście* z punktu pierwszego).
  - Jeżeli wartość *exchangedMoney* obecnego węzła (węzła początkowego) nie jest większa od wartości podanej przez użytkownika i jest ostatnią walutą na *ArrayLiście*, oznacza to, że dla otrzymanych danych nie można znaleźć arbitrażu. Komunikat o tym zostaje wyświetlony.
- (b) Jeśli obecny węzeł jest węzłem początkowym, a flaga zmiany kosztu jest zapalona, to należy powtórzyć punkty 5-7, ponieważ jeszcze nie nadano najkorzystniejszych kosztów wszystkim węzłom.
- (c) Jeśli w obecnym węźle wartości *cost* i *charge* są liczbami ujemnymi, a *previous* ma wartość *null*, to znaczy, że jeszcze nie wiemy jak dotrzeć do tego węzła i bez aktualizacji kosztów węzłów sąsiadujących z obecnym węzłem powtarzamy punkt 7.
- (d) W pozostałym wypadku (różnym od a, b, c i d) powtarzamy punkty 6-7.

### 3.3.3 Wyświetlanie ścieżki

Informacje o elementach szukanej ścieżki znajdują się w polach *previous* obiektów *Currency*. Napis zawierający całą ścieżkę tworzy się dzięki przechodzenie grafu szlakiem referencji do walut zawartych w polach *previous*:

- w przypadku znalezienia korzystnej wymiany walut należy rozpocząć od waluty docelowej i podążać kolejno referencjami z pól *previous*, odkładając na stos kolejne nazwy walut, aż dotrzemy do waluty początkowej,
- w przypadku znalezienia dowolnego arbitrażu, należy rozpocząć od waluty, która została odkryta jako możliwość arbitrażu, a następnie podążając kolejno referencjami z pól *previous*, odkładać na stos kolejne nazwy walut, aż dotrzemy z powrotem do danej waluty.

Po tych operacjach należy pobierając ze stosu nazwy walut stworzyć napis reprezentujący korzystną ścieżkę wymiany walut lub arbitraż. Do realizacji tego zadania przeznaczona jest klasa *Path*, która będzie zawierać potrzebne metody, a także będzie zwracana przez metody odpowiedzialne za znajdowanie korzystnej ścieżki wymiany waluty oraz znajdowanie dowolnego arbitrażu.

---

## 4 Opis ważniejszych metod

- Path **findExchange**(waluta początkowa, waluta docelowa)

Metoda wykonuje działania opisane w części 3.3.1 tego dokumentu.

- Path **findArbitrage**(kwota)

Metoda wykonuje działania opisane w części 3.3.2 tego dokumentu.

- String **makePath**(stos walut)

Metoda wykonuje działania opisane w ostatnim akapicie części 3.3.3 tego dokumentu.

---

---

## 5 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.
- 

## 6 Informacje o sprzęcie i oprogramowaniu

---