Specyfikacja implementacyjna

1. Informacje ogólne

   1.1. Forma przekazu
   Program wykonany jest w formie aplikacji działającej w terminalu. Do włączenia go wymagane jest posiadanie kompilatora Java. Program nie posiada dedykowanego GUI.

   1.2. Sposób uruchomienia
   Aby włączyć program należy przejść w terminalu do katalogu zawierającego pilk wykonywalny **VaccOpt.jar** i włączyć go wpisując komendę *java -jar VaccOpt.jar "nazwaPliku"*, gdzie nazwaPliku to nazwa pliku z danymi. W przypadku wywołania bez argumentów program domyślnie korzysta z pliku *dane.txt*.

2. Opis pakietów
   Pakiety zostały podzielone ze względu na funckjonalność.

   2.1. Pakiet load
   Głównym zadaniem tego pakietu jest wczytywanie danych z pliku. Klasa zawarta w tym pakiecie sprawdza, czy została podana nazwa pliku z danymi (jeżeli nie to otwiera plik "dane.txt"). Następnie sprawdza, czy format danych w pliku się zgadza.Dane w pliku muszą być oddzielone od siebie separatorem "|", a nazwy aptek i dystrybutorów nie mogą zawierać tego znaku. Każda z aptek musi mieć połączenie z każdym z dystrybutorów. Podaż musi być większa lub równa popytowi. Wszystke liczby zawarte w pliku muszą być liczbami dodatnimi, a zapotrzebowanie na szczepionki jak i ich dostępna ilość muszą być liczbami całkowitymi. Identyfikatory aptek muszą być unikatowe, tak samo musi być z identyfikatorami dystrybutorów, jednak zarówno apteka jak i dystrybutor mogą mieć to samo id (np. 0), ponieważ są innego typu. Możliwe informacje zwrotne o błędach:
   - dane są źle sformatowane, separator "|" nie oddziela kolumn, lub występuje w nazwie apteki bądź dystrybutora
   - co najmniej jedna z aptek nie jest połączona ze wszystkimi dystrybutorami,
   - w pliku występują liczby ujemne,
   - nie można potrzebować fragmentu szczepionki,
   - podaż mniejsza od popytu,

- dublujące się ID apteki lub dystrybutora.
Na koniec dane są zapisywane do dwóch macierzy (tablic dwuwymiarowych). Jedna zawiera informacje o połączeniach aptek z dystrybutorami, druga o zapotrzebowaniu aptek i podaży dystrybutorów.
Przykładowy wygląd pliku danego jako input:

```
# Producenci szczepionek (id | nazwa | dzienna produkcja)
0 | BioTech 2.0 | 900
1 | Eko Polska 2020 | 1300
2 | Post-Covid Sp. z o.o. | 1100
# Apteki (id | nazwa | dzienne zapotrzebowanie)
0 | CentMedEko Centrala | 450
1 | CentMedEko 24h | 690
2 | CentMedEko Nowogrodzka | 1200
# Połączenia  (id prod | id apt | max liczba szczepionek | koszt sztuki)
0 | 0 | 800 | 70.5
0 | 1 | 600 | 70
0 | 2 | 750 | 90.99
1 | 0 | 900 | 100
1 | 1 | 600 | 80
1 | 2 | 450 | 70
2 | 0 | 900 | 80
2 | 1 | 900 | 90
2 | 2 | 300 | 100
```

2.2. Pakiet aprox

Ten pakiet zawiera klasy, w których rozwiązywany jest trzon zadania oraz klasę main zawierającą tylko metodę main(). Ten pakiet spełnia dwa zadania:
- rozwiązuje główny problem przedstawiony w zadaniu metodą VAM (metoda aproksymacji Vogla),
- zawiera klasę, w której znajduje się metoda main(), w której wywoływane są najważniejsze metody pozostałych pakietów.
Tak naprawdę ten pakiet jest spoiwem łączącym cały program w jedną, działającą całość.

2.3. Pakiet save

W tym pakiecie znajduje się klasa, której zadaniem jest stworzenie i sformatowanie pliku *result.txt*. W tym pliku znajdują się informacje, które są wynikiem działań w pakiecie approx. Przykładowa struktura pliku *result.txt* jest przedstawiona poniżej.

```
BioTech 2.0     -> CentMedEko Centrala [Koszt = 300 * 70.5 = 21150 zł]
Eko Polska 2020 -> CentMedEko Centrala [Koszt = 150 * 100 = 15000 zł]
/*
...
pozostałe ustalone połączenia pomiędzy producentami a aptekami
...
*/
Opłaty całkowite: 36150 zł
```

3. Opis klas

3.1. Loader

Klasa ta jest częścią pakietu *load*. Nie dziedziczy po żadnej klasie, nie implementuje też żadnych interfejsów. Importuje java.io.FileNotFoundException, java.io.FileReader oraz java.io.IOException. Zadaniem tej klasy jest zapisanie danych z pliku do utworzonych macierzy.
W konstruktorze tej klasy wywołuję metodę tworzące FileReadera i zapisującą informacje z niego pozyskane w postaci macierzy.
W klasie znajdują się 2 zmienne będące macierzami.
Metoda read
Nagłówek: public void read(string fileName)
Zadanie: tworzy obiekt klasy FileReader i przypisuje go do odpowiedniego pliku podanego jako argument tej metody, następnie wywołuje metodę ustawiającą dane w macierzach.
Co zwraca: Funkcja typu void, czyli nic nie zwraca.
Metoda getConnections
Nagłówek: public int[][] getConnections()
Zadanie: getter macierzy z połączeniami
Co zwraca: macierz z połączeniami
Metoda getDaS
Nagłówek: public int[][] getDaS()
Zadanie: getter macierzy popytem i podażą
Co zwraca: macierz z popytem i podażą

3.2. Aprox

Klasa jest trzonem pakietu *aprox*.Nie dziedziczy po żadnej klasie, nie implementuje też żadnych interfejsów. Zadaniem tej klasy jest operowanie na utworzonych macierzach w taki sposób, aby korzystając z pomocy trzeciej macierzy zoptymalizować proces zakupu

szczepionek i wybrać jak najmniejszą cenę.
W konstrukorze tej klasy korzystam z dwóch macierzy przekazywanych jako argumenty wywołania w celu wywołania metody tworzącej trzecią macierz oraz wywołuję metodę aproksymującą.
Metoda remainingDaS
Nagłówek: public int[][] remainingDaS(int[][] connections, int[][] das)
Zadanie: stworzenie macierzy z pozostałym zapotrzebowaniem i dostępnymi szczepionkami
Co zwraca: macierz z pozostałym zapotrzebowaniem i dostępnymi szczepionkami
Metoda vam
Nagłówek: public int[][] vam(int[][] connections, int[][] das)
Zadanie: stworzenie macierzy obrazującej ile szczepionek od każdego z dystrybutorów zostało sprzedanych danej aptece. Wykorzystany będzie algorytm VAM.
Algorytm: VAM polega na kolejnym znajdowaniu różnicy między dwoma najmniejszymi wartościami w każdej kolumnie i każdym wierszu. Następnie wybierana jest największa liczba spośród tych różnic. Jeżeli liczba ta jest różnicą pochodzącą wartości w jednej kolumnie, to oznacza to, że w tej kolumnie zostanie zawarta pierwsza transakcja. Jeżeli transakcja wyczerpuje podaż lub popyt, to wtedy do dalszych akcji nie jest brany pod uwagę dany producent(dostawca) lub apteka. Akcja ta jest powielana do momentu zospokojenia potrzeb wszystkich aptek(odbiorców). Co zwraca: macierz obrazującą, który producent sprzedał ile szczepionek danej aptece.

3.3. Main
Klasa ta należy do pakietu *aprox*. Nie dziedziczy po żadnej klasie, nie implementuje interfejsów. Jedynym zadaniem tej klasy jest posiadanie metody main, w której tworzone są obiekty pozostałych klas i wywoływane są ich metody.

3.4. ResultsSaver
Klasa ta należy do pakietu *save*. Nie dziedziczy po żadnej klasie, nie implementuje interfejsów. Zadaniem tej klasy jest stworzenie pliku *result.txt*, a następnie zapisanie do niego danych. Importuje java.io.FileWriter oraz java.io.IOException. W konstruktorze tworzę nową instancję obiektu FileWriter z argumentem *result.txt*.

4. Testowanie

   4.1. Użyte narzędzia Testy wykonałem używając JUnit 4.

   4.2. Konwencja testów Nazwy testów sugerują, co powinno być zwracane w danej metodzie. Przykładem jest shouldRaiseWrongTextFormatError.

   4.3. Warunki brzegowe Zdecydowanie najwięcej błędów może występować przy wczytywaniu danych z pliku. Tutaj będzie najwięcej testów, jeden test dla każdego typu błędu. Testy będą też dotyczyły samej aproksymacji VAM.