

# Specyfikacja implementacyjna projektu indywidualnego

## AiSD GR1

Hubert Nakielski

Listopad 2020

### Informacje ogólne

Program napisany będzie w języku Java 14.0.1 i udostępniony jako plik o nazwie VaccineOptimizer w formacie .jar.

Należy go uruchomić przez komendę: `java -jar VaccineOptimizer.jar`. Program następnie poprosi o podanie ścieżki pliku wejściowego. Plik wyjściowy zostanie utworzony w folderze result.

### Opis modułów

#### Pakiet *vaccine*

Pakiet zawierający wszystkie pakiety z kodem źródłowym.

#### Pakiet *vaccine.file*

Odpowiedzialny za wszystkie czynności związane z plikami wejściowymi i wyjściowymi.

#### Pakiet *vaccine.calculations*

Odpowiada za liczenie najtańszej konfiguracji zakupionych szczepionek.

#### Pakiet *vaccine.objects*

Zawiera klasy odpowiedzialne za tworzenie obiektów takich jak: Producent, Apteka, Połączenie (między apteką, a producentem)

#### Folder *vaccine.result*

W tym miejscu będzie zapisywany plik wyjściowy

## Opis klas

### Klasa *ConfigurationIO*

Zawiera się w module *file*, czyta plik wejściowy, tworzy listy aptek, producentów i połączeń. Tworzy plik wyjściowy z konkretną konfiguracją. Zawiera 3 zmienne, 2 metody dostępne i jedną metodę modyfikującą:

- `path`: `String`
  - `manufacturerList`: `List<Manufacturer>`
  - `pharmacyList`: `List<Pharmacy>`
- 
- ▶ `getManufacturerList()`: `List<Manufacturer>`
  - ▶ `getPharmacyList()`: `List<Pharmacy>`
  - ▶ `setPath()`: `void`

Metody w klasie:

#### **loadFromFile(String filePath)**

Metoda odpowiedzialna za czytanie pliku wejściowego.

Wartość zwracana: `void`

#### **isPharmaciesInfo(String line)**

Sprawdza czy pod sprawdzaną linijką znajdują się informacje o aptekach.

Wartość zwracana: `boolean`

#### **isManufacturersInfo(String line)**

Sprawdza czy pod sprawdzaną linijką znajdują się informacje o producentach.

Wartość zwracana: `boolean`

#### **isConnectionsInfo(String line)**

Sprawdza czy pod sprawdzaną linijką znajdują się informacje o połączeniach.

Wartość zwracana: `boolean`

#### **parseManufacturersLine(String line)**

Dodaje producentów i ich dane do listy zgodnie z plikiem wejściowym.

Wartość zwracana: `void`

#### **parsePharmaciesLine(String line)**

Dodaje apteki i ich dane do listy zgodnie z plikiem wejściowym.

Wartość zwracana: `void`

#### **parseConnectionsLine(String line)**

Dodaje połączenia do listy zgodnie z plikiem wejściowym.

Wartość zwracana: `void`

**saveToFile(List<Pharmacy> pharmacyList)**

Metoda odpowiedzialna za wpisywanie gotowej konfiguracji do pliku wyjściowego.

Wartość zwracana: void

### **Klasa *Manufacturer***

Występuje w module *object*. Zawiera 5 zmiennych, ich metody dostępne oraz jedną metodę modyfikującą:

- id: int
  - name: String
  - daily\_production: int
  - connectionList: List<Connection>
  - vamFactor: int
- 
- ▶ getId(): int
  - ▶ getName(): String
  - ▶ getDailyProduction(): int
  - ▶ getConnectionList(): List<Connection>
  - ▶ getVamFactor(): int
- 
- ▶ setVamFactor(int vamFactor): void

### **Klasa *Pharmacy***

Występuje w module *object*. Zawiera 5 zmiennych, ich metody dostępne oraz jedną metodę modyfikującą:

- id: int
  - name: String
  - need: int
  - connectionList: List<Connection>
  - vamFactor: int
- 
- ▶ getId(): int
  - ▶ getName(): String
  - ▶ getNeed(): int
  - ▶ getConnectionList(): List<Connection>
  - ▶ getVamFactor(): int
- 
- ▶ setVamFactor(int vamFactor): void

Dodatkowo klasa ta zawiera metodę:

**addConnection(Manufacturer manufacturer, Pharmacy pharmacy, int quantity, double price)**

Dodaje połączenie do listy połączeń (używana jest podczas czytania pliku)

Wartość zwracana: void

## Klasa *Connection*

Występuje w module *object*. Zawiera 4 zmienne, ich metody dostępne oraz jedną metodę modyfikującą:

- **manufacturer:** Manufacturer
  - **pharmacy:** Pharmacy
  - **quantity:** int
  - **price:** double
- 
- ▶ **getManufacturer:** Manufacturer
  - ▶ **getPharmacy():** Pharmacy
  - ▶ **getQuantity():** int
  - ▶ **getPrice():** double
- 
- ▶ **setQuantity(int quantity):** void

## Klasa *VAM*

Zawiera się w module *calculations*, liczy konfigurację o najmniejszym koszcie używając metody VAM (Vogel's approximation method). Zawiera 4 zmienne oraz konstruktor:

- **configurationIO:** ConfigurationIO
  - **manufacturerList:** List<Manufacturer>
  - **pharmacyList:** List<Pharmacy>
  - **connectionList:** List<Connection>
- 
- ▶ **Pharmacy(int id, String name, int need)**

Metody w klasie:

**minimizeCost(List<Pharmacy> pharmacyList, List<Manufacturer> manufacturerList)**

Metoda odpowiedzialna za liczenie minimalnego kosztu.  
Wartość zwracana: List<Connection>

**calculateVAMFactor()**

Metoda odpowiedzialna za obliczenie współczynnika VAM dla każdej apteki i dla każdego producenta.  
Wartość zwracana: void

**findGreatestVAMFactor()**

Znajduje najwyższy współczynnik VAM i zwraca daną aptekę lub producenta, w której on występuje.  
Wartość zwracana: Object

### **adjustPossibleQuantity(Pharmacy pharmacy, Manufacturer manufacturer)**

Ustala najwyższą możliwą ilość szczepionek dostarczanej z danego producenta do danej apteki, uwzględniając:

- zapotrzebowanie apteki,
- dzienną produkcję producenta,
- dzienną maksymalną liczbę dostarczanych szczepionek od danego producenta do danej apteki ( wynikające z umowy )

Wartość zwracana: **int**

### **generateConfigurationToFile()**

Wywołuje metodę *saveToFile(List<Pharmacy>)* z klasy ConfigurationIO podając przy tym gotową listę aptek **connectionList** ( zawierającą listę połączeń w odpowiedniej już konfiguracji)

Wartość zwracana: **void**

## **Logika liczenia najtańszej konfiguracji**

Liczenie odbywać się będzie używając metody VAM. Dla tej metody utworzymy dodatkowe zmienne (współczynnik VAM) dla każdego obiektu ( apteka / producent ).

### **1. Liczę współczynnik VAM**

- 1.1 dla każdego obiektu sprawdzam cenę poszczególnego połączenia, które jest związane z danym obiektem)
- 1.2 wybieram minimum z tych cen;
- 1.3 wybieram drugie minimum z tych cen;
- 1.4 obliczam różnicę z tych dwóch minimów.

### **2. Znajduję największy współczynnik**

- 2.1 wybieram maximum ze współczynników wszystkich obiektów;
- 2.2 zapamiętuję aptekę, dla której ten współczynnik wystąpił.

### **3. Ustalam najtańszą konfigurację dla podanej apteki**

- 3.1 dla zapamiętanej apteki znajduję połączenie, dla którego cena szczepionki będzie najniższa i zapamiętuję je;
- 3.2 ustalam najwyższą ilość szczepionek dla zapamiętanego połączenia, spełniającą warunki:
  - suma kupionych przez aptekę szczepionek nie przekracza dziennego zapotrzebowania
  - suma sprzedanych przez producenta szczepionek nie przekracza dziennej produkcji
  - ilość kupionych/sprzedanych szczepionek w danym połączeniu nie przekracza dziennej maksymalnej liczby dostarczanych szczepionek przez producenta do apteki
- 3.3 dla tej samej apteki znajduję drugie najtańsze połączenie, dla którego cena szczepionki będzie najniższa i zapamiętuję je;
- 3.4 ponownie wykonuję punkt 3.2;
- 3.5 kończę ustalanie najtańszej konfiguracji dla tej apteki, gdy suma kupionych przez aptekę szczepionek będzie równa dziennemu zapotrzebowaniu.

### **4. Wracam do punktu 1. bez uwzględniania już zapełnionych aptek**

4.1 kończę, gdy każda apteka ma już ustaloną konfigurację

## **Testowanie**

DODAJ

## **Użyte narzędzia**

DODAJ

## **Konwencja**

DODAJ

## **Warunki brzegowe**

DODAJ

# Diagram klas

