Wstęp

W naszym internetowym supermarkecie chcemy rozszerzyć asortyment produktów, które sprzedajemy o artykuły niespożywcze, takie jak elektronika czy przedmioty do domu. Charakteryzują się one tym, że często nie da się ich dostarczyć naszym standardowym samochodem dostawczym, ponieważ są one często po prostu za duże. Drugim powodem bywa to, że będziemy również oferować przedmioty, które znajdują się u zewnętrznych dostawców i nie możemy ich dostarczać naszą flotą. W obu takich przypadkach, zakupy klienta trzeba wysłać specjalistycznym kurierem. Z drugiej strony, oczywistym jest to, że kurierzy nie mogą dostarczać wszystkich przedmiotów które sprzedajemy, w szczególności produktów spożywczych.

W związku z tym, że nie wszystkie produkty da się wysłać wszystkimi sposobami dostawy, musimy podzielić przedmioty w internetowym koszyku klienta na grupy dostaw. Zadanie polega na optymalnym podziale przedmiotów w koszyku tak, aby zminimalizować liczbę wymaganych dostaw. Po wyznaczeniu minimalnej liczby grup dostaw, należy upewnić się, że będziemy tworzyć największe możliwe grupy dostaw (patrz rysunek 1.). Zaobserwowaliśmy, że klienci często decydują się tylko na zakup przedmiotów z największej grupy dostawy, pozostawiając resztę w koszyku.

Zadanie

Twoim zadaniem jest stworzenie biblioteki, która podzieli przedmioty w koszyku klienta na grupy dostaw. Zdefiniowaliśmy już API, które chcielibyśmy wykorzystać w naszym programie - patrz “Struktura programu i testowanie”. W celu poprawnego działania, biblioteka musi wczytać plik konfiguracyjny, który zawiera możliwe sposoby dostawy wszystkich oferowanych w sklepie produktów. Jako że ta konfiguracja nie zmienia się często, przechowujemy ją w pliku, który ma zostać przeczytany przez Twoją implementację biblioteki.

Dane wejściowe

Program ma podzielić produkty znajdujące się w koszyku. Zostaną one przekazane do API biblioteki w formie listy z nazwami produktów:

```
[
 "Steak (300g)",
 "Carrots (1kg)",
 "Soda (24x330ml)",
 "AA Battery (4 Pcs.)",
 "Espresso Machine",
 "Garden Chair"
]
```

Przykładowy parametr algorytmu (`List<String> items`)

Do przeprowadzenia podziału potrzebna jest także definicja możliwych sposobów dostawy produktów oferowanych w naszym sklepie. Ponieważ konfiguracja sposobu dostawy produktów nie zmienia się często, przechowujemy ją w pliku konfiguracyjnym. W związku z tym, chcemy do algorytmu przekazać absolutną ścieżkę do pliku konfiguracyjnego w formacie JSON. Kluczem w mapie jest nazwa produktu, a wartością - lista z możliwymi sposobami dostawy danego produktu:

```
{
 "Carrots (1kg)": ["Express Delivery", "Click&Collect"],
 "Cold Beer (330ml)": ["Express Delivery"],
 "Steak (300g)": ["Express Delivery", "Click&Collect"],
 "AA Battery (4 Pcs.)": ["Express Delivery", "Courier"],
 "Espresso Machine": ["Courier", "Click&Collect"],
 "Garden Chair": ["Courier"]
}
```

Przykładowa zawartość pliku config.json, do którego ścieżka będzie przekazana w `absolutePathToConfigFile`

Ograniczenia

- W katalogu produktów (pliku konfiguracyjnym) znajduje się:
    - Maksymalnie 1000 produktów;
    - Do 10 różnych sposobów dostaw
- W koszyku klienta znajduje się maksymalnie 100 produktów

Spodziewany wynik

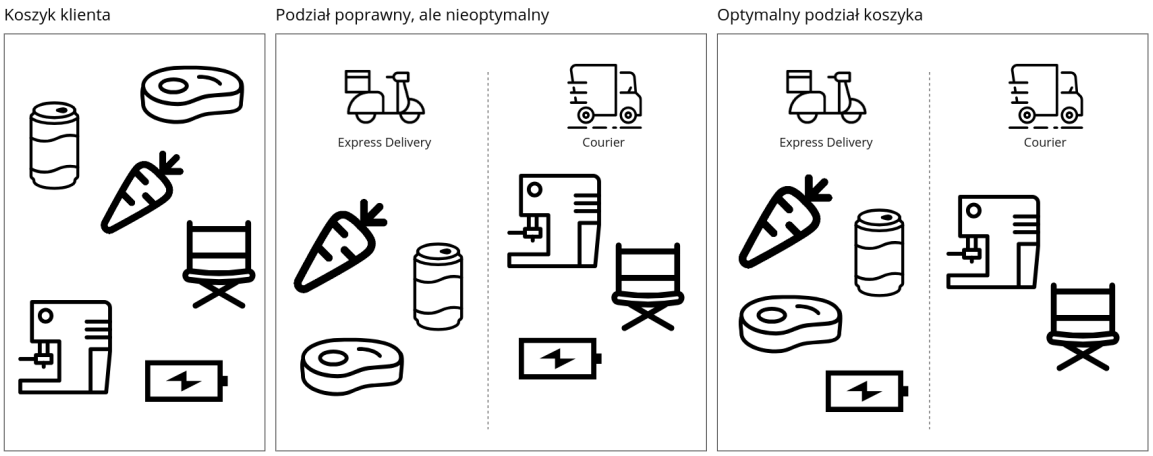
Algorytm w wyniku ma zwrócić podział produktów na grupy dostawy jako mapę. Jej kluczem ma być sposób dostawy, a wartością lista produktów. Chcielibyśmy, aby:

- Algorytm dzielił produkty na możliwie minimalną liczbę grup dostaw.
- Największa grupa zawierała możliwie najwięcej produktów (patrz rys. 1.)

Reprezentacja wyniku w postaci JSON-a:

```
{
  "Express Delivery": ["Steak (300g)", "Carrot (1kg)", "Cold Beer (330ml)", "AA Battery (4 Pcs.)"],
 "Courier": ["Espresso Machine", "Garden Chair"]
}
```

Uwaga: biblioteka nie musi nigdzie pokazywać wyniku w taki sposób. Wystarczy, że będzie zwracała odpowiednią mapę z wywołania funkcji `split(...)` (patrz “Struktura programu i testowanie”)



Rysunek 1

Struktura programu i testowanie

Aby ułatwić zadanie, zdefiniowaliśmy typ “`BasketSplitter`”, którego implementację musisz uzupełnić. Dostarczoną przez Ciebie bibliotekę będziemy sprawdzać poprzez tworzenie instancji typu BasketSplitter, przekazując absolutną ścieżkę do pliku z konfiguracją. Następnie będziemy uruchamiać serię testów wywołujących publiczną metodę split(...) uruchamiając algorytm dla różnych koszyków.

```
package com.ocado.basket;

import java.util.List;
import java.util.Map;
/* ... */
```

```
public class BasketSplitter {

    /* ...  */

    public BasketSplitter(String absolutePathToConfigFile) {
        /* ...  */
    }

    public Map<String, List<String>> split(List<String> items) {
        /* ...  */
    }

    /* ...  */
}
```

Nasze oczekiwania

- W odpowiedzi na maila z zadaniem spodziewamy się archiwum .zip lub .tar.gz z:
    - kodem źródłowym aplikacji
    - plikiem .jar ze zbudowaną biblioteką
- Można korzystać z dowolnych bibliotek, należy jednak wiedzieć, co robią.
    - Uwaga! Prosimy aby jar zawierał wszystkie wymagane zależności aplikacji (tzw. fat-jar).
- Aplikacja powinna być napisana w Javie 17 lub 21
- To, na co zwracamy uwagę:
    - poprawność wyników
    - czytelność kodu
    - jakość projektu - chcielibyśmy, aby kod, a przynajmniej jego kluczowe fragmenty, były pokryte testami
- Możesz uzyskać dodatkowe punkty za użycie narzędzia do budowania projektu oraz stworzenie pliku README wraz z kluczowymi informacjami.