Sales Manager

Specyfikacja implementacyjna

Krzysztof Głodowski
230331
Łukasz Krok
230349
14 grudnia 2010

DIAGRAM KLAS

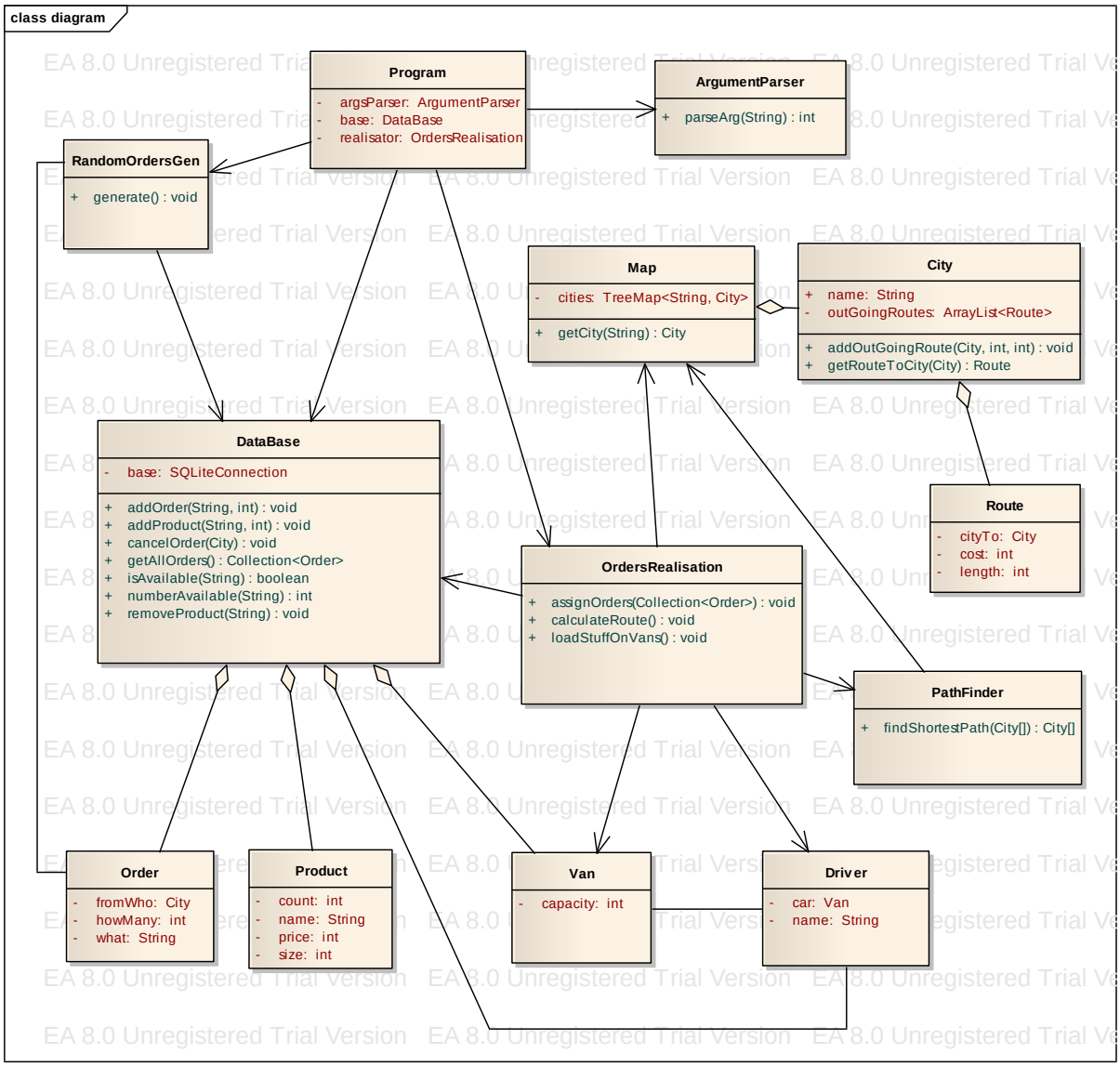

Diagram klas programu

OPIS KLAS

Program

Klasa program zarządza kolejnym wykonaniem czynności. Na początku analizuje argumenty wywołania i na ich podstawie wykonuje określone zadanie. Następnie używa odpowiedniego obiektu do realizacji tego zadania i czeka na wynik zwrotny, który wyświetla na ekranie

ArgumentParser

Jest używany przez klasę program do analizy pobranych argumentów wywołania. Każda dostępna opcja w programie jest reprezentowana przez liczbę naturalną. W zależności od tego jaki argument został otrzymany, do programu zostaje zwracana przypisana mu liczba. W ten sposób program „wie" co rozkazano mu robić.

RandomOrdersGen

Klasa zajmuje się losową generacją zamówień, które są zapisywane w bazie.

DataBase

Klasa jest pośrednikiem między programem a bazą danych SQLite. Zawiera w sobie metody pozwalające na zapis do tej bazy, modyfikację oraz odczyt. Pozwala pobrać wszystkie złożone zamówienia oraz sprawdzić liczbę dostępnych produktów i czy dany produkt znajduje się w bazie.

Order

Klasa jest reprezentacją zamówienia. Posiada takie informacje jak nazwę produktu, zamówioną ilość i miejsce skąd zamówienie wpłynęło.

Product

Reprezentacja produktu. Ma swoją nazwę, cenę, dostępną ilość a także rozmiar jednej sztuki. Rozmiar ma kluczowe znaczenie przy pakowaniu do samochodu

Map

Klasa zawiera w sobie zbiór miast i dróg tworząc mapę. Pełni rolę kontenera

City

Reprezentacja miasta. Posiada swoją nazwę oraz listę wychodzących dróg z tego miasta. Możliwe jest dodanie nowej drogi a także otrzymanie drogi, która prowadzi do szukanego przez nas miasta (coś jak drogowskazy☺).

Route

Reprezentacja drogi. Posiada długość, koszt oraz wskazanie na miasto, do którego prowadzi. Obiekt tej klasy jest drogą w jednym kierunku. Aby uzyskać drogę dwu kierunkową, tworzymy drugi obiekt wskazujący w przeciwną stronę.

OrdersRealisation

Klasa zajmuje się przydzielaniem zamówień kierowcom. Robi to w ten sposób, że najpierw pobiera wszystkie zamówienia, potem dzieli je między kierowców, ale robi to tak, by każdy z kierowców nie musiał zwiedzić dwóch skrajnych krańców mapy. Chodzi o to, aby każdy z kierowców rozwoził towar w pobliskich miastach a nie tych oddalonych od siebie bardzo daleko. Następnie dla każdego kierowcy wyznaczana jest najkrótsza trasa pomiędzy przydzielonymi miastami. Kolejnym etapem jest zapakowanie towaru. Na końcu przestrzeni ładunkowej samochodu znajduje się ten towar, który zostanie dowieziony ostatni. Unikamy w ten sposób sytuacji, gdy kierowca dojeżdża do pierwszego klienta a towar dla niego jest na końcu i musi wszystko poprzewalać, żeby się do niego dostać. Po tych wszystkich czynnościach do programu głównego zostaje wysyłany raport sprzedaży.

PathFinder

Klasa jest używana przez *OrdersRealisation* do wyszukiwania najkrótszej kombinacji połączeń między miastami. Używany jest do tego algorytm genetyczny.

Driver

Klasa reprezentuje kierowcę. Ma on swoje nazwisko i przypisany samochód.

Van

Reprezentacja samochodu dostawczego z jednym polem oznaczającym pojemność przestrzeni towarowej.

ZASTOSOWANE ALGORYTMY

Wyznaczanie najbardziej korzystnej drogi pomiędzy miastami

Do wyznaczenia permutacji połaczeń między miastami skorzystamy z algorytmu genetycznego. Aby użyć tego algorytmu musimy wybrać operator krzyżowania. Operator OX tworzy potomka poprzez wybór podtrasy z jednego rodzica i pozostawienie wzajemnego uporządkowania miast z drugiego.

Czyli dwie trasy (liczby reprezentują miasta):

```
r1 = (1 2 3 [4 5 6 7] 8 9),

r2 = (4 5 2 [1 8 7 6] 9 3)
```

dają potomków po wymianie podtras miast:

```
p1 = (x x x [4 5 6 7] x x),

p2 = (x x x [1 8 7 6] x x).
```

Następnie z r2 tworzymy ciąg miast, zaczynając od pierwszego za drugim cięciem (czyli znakiem]): 9, 3, 4, 5, 2, 1, 8, 7, 6, wykreślamy te, które już znajdują się w p1 otrzymując: 9, 3, 2, 1, 8, po czym wstawiamy je od pierwszego x za drugim cięciem:

```
p1 = (2 1 8 [4 5 6 7] 9 3),

p2 = (3 4 5 [1 8 7 6] 9 2).
```

(w przypadku drugiego potomka postępowanie wygląda tak samo)

Krzyżowanie będziemy wykonywać do tej pory, aż nie znajdziemy odpowiedniej ilości permutacji. Wtedy wybieramy kombinację o najkrótszej trasie i najmniejszym koszcie przejazdu.

Algorytm przydzielania zadań kierowcom

Algorytm ma za zadanie tak podzielić zamówienia między kierowców, żeby każdy z nich miał stosunkowo małą powierzchnię mapy do przejechania. Weźmy na przykład czterech kierowców i cztery województwa: zachodnio-pomorskie, mazowieckie, dolnośląskie i podkarpackie. Chodzi o to, żeby jeden kierowca nie musiał jechać z mazowieckiego do dolnośląskiego a drugi z dolnośląskiego do podkarpackiego itd… Algorytm musi tak przydzielić zamówienia, żeby każdy z czterech kierowców poruszał się po jak najmniejszym obszarze, tu tylko w województwie.

Algorytm załadunku samochodu

Algortym zajmuje się układaniem towaru w samochodzie. Robi to w ten sposób, że towar dla odbiorcy do którego kierowca dojedzie najpierw, znajduje się na samym początku. Na samym końcu znajduje się towar klienta, do którego kierowca dotrze ostatni.

STRUKTURA PLIKÓW

Przykładowy plik z listą produktów do wczytania:

```
Dezodorant  1000
Krem         500
Szampon     2500
Perfumy      800
Lakier       700
```

Przykładowy plik zamówienia do wczytania: (cel poprzedza się znakiem '#')

```
#Warszawa
Dezodorant   200
Krem         150
Perfumy      500
#Łódź
Szampon     1000
Lakier       400
```

Przykładowy plik mapy:

```
Cities:
Warszawa
Łódź
Gdańsk
Wrocław
Kraków
Routes:
Warszawa Łódź    120  40
Warszawa Gdańsk  310 120
Gdańsk Wrocław   450 180
Łódź Kraków      160  65
Kraków Wrocław   100  40
Wrocław Warszawa 200 130
```

Pod etykietą Cities wypisane są miasta na mapie. Pod etykietą Routes znajdują się drogi. Początek drogi to pierwsze miasto, koniec drugie. Trzecie pole to długość drogi, czwarte to koszt.

ZESTAWIENIE KLAS I ICH METOD

Klasa	Metoda	Opis
DataBase	`void addOrder(String what, int howMany);`	Dodaje do bazy zamówienie na towar o nazwie what i ilość howMany.
	`void addProduct(String name, int count);`	Dodaje product o nazwie name i ilości count.
	`Collection<Order> getAllOrders();`	Zwraca wszystkie zamówienia złożone w bazie.
	`void cancelOrder(City from);`	Anuluje zamówienie z danego miasta.
	`void removeProduct(String name);`	Usuwa wszystkie przedmioty o danej nazwie.
	`void decreaseCount(String name, int n);`	Zmniejsza liczbę dostępnych produktów o nazwie name o liczbę n.
	`boolean isAvailable(String name);`	Sprawdza czy dany produkt jest dostępny.
	`int countAvailable(String name);`	Zwraca liczbę dostępnych produktów o nazwie name.

RandomOrdersGen	`void generate(int n);`	Generuje losowo n różnych zamówień.
ArgumentParser	`int parseArg(String arg);`	Rozpoznaje podany argument i zwraca liczbę reprezentującą oczekiwaną czynność. Stałe liczbowe są zapisane w klasie ArgumentParser.
Map	`City getCity(String name);`	Zwraca obiekt miasta o podanej nazwie.
City	`void addOutGoingRoute(City dest, int length, int cost);`	Dodaje wychodzącą drogę prowadzącą do dest o długości length i koszcie cost.
	`Route getRouteToCity(City c);`	Zwraca drogę prowadzącą do wskazanego miasta.
OrdersRealisation	`void assignOrders (Collection<Order>);`	Implementacja algorytmu przydzielania zadań.
	`void calculateRoute();`	Metoda znajduje na mapie najkrótszą trasę przejazdu między miastami, gdzie trzeba dowieźć towar z zamówień. Dla każdego kierowcy oddzielnie.
	`void loadStuffOnVans();`	Implementacja algorytmu załadunku towaru do samochodów dostawczych.
	`void generateReport();`	Metoda generuje raport sprzedaży do pliku html.
PathFinder	`City[] findShortestPath(City[] c);`	Metoda wyznacza możliwie jak najwięcej kombinacji tras między podanymi miastami i zwraca najkrótszą ułożoną w tablicy.