Programowanie Obiektowe
Pliki
Zadanie oceniane nr 3a
26-05-2022

W dniu dzisiejszym komponenty do zadania znajdują się w Państwa repozytorium (katalog zadanie_oceniane_3a/resources). Po zakończeniu pracy konieczne jest wgranie zmian do repozytorium (add + commit + push) w katalogu o nazwie w stylu: zadanie_oceniane_3a. Fakt wgrania plików do swojego repozytorium można sprawdzić samodzielnie logując się (via www) na swoje konto i sprawdzając czy pojawiły się tam wszystkie zmiany.

"Odloty..."

... i przyloty – to zagadnienie będzie tematem wiodącym dzisiejszego zadania. Tradycyjnie, należy wczytać zawartość pliku zawierającego informacje o ruchu w pewnym porcie lotniczym położonym tylko kilka kilometrów od naszej bieżącej lokalizacji.

Prace do wykonania:

1. **Zastanowić się nad naturą danych, które przyjdzie Państwu parsować.**
   Plik loty.txt zawiera dwa rodzaje encji z których każda zajmuje jedną linijkę. Poszczególne jej składowe oddzielone są sekwencją znaków: ###.

   Wiersz opisujący odlot ma następujących schemat:

   O (odlot)###godzina###dokąd?###id lotu###linia
   ###status (started/boarding/canceled)###check in (zakres nr stoisk check-in: od-do)###

   Przykład: O###15:50###Ulm (FMM)###W6 1475###Wizz Air###started###108-118###

   Wiersz opisujący przylot ma następujący schemat:

   P (przylot)###godzina###skąd?###id lotu###linia###wylądował?###nr wyjścia###

   Przykład: P###18:55###Paryż (CDG)###AF 1046###Air France###true###1###

2. **Model danych**
   Stworzyć klasy których instancje odzwierciedlają wiersze danych zapisanych w pliku (pamiętając że przylot to nie to samo co odlot – to dwa różne typy). Dla każdej wyizolowanej z danego wiersza danej (cel, linia, itp.) powinno w danej klasie być stworzone pole odpowiedniego typu. Niech już będzie że godzina jest Stringiem, a nie dedykowanym typem. Trzeba mieć na uwadze, żeby obiekty klas opisujących przyloty i odloty można było wrzucić do jednej kolekcji. Znaczniki "O" i "P" wskazują typ, więc nie trafiają do obiektów.

   Należy pamiętać że:
   - Zakładamy że struktura danych w pliku i ich typy **są poprawne**
   - To czy dany obiekt danego typu jest **równy** drugiemu zależy od tego czy ich typy oraz id lotu są takie same. Zatem trzeba uwzględnić to kryterium w dobrze znany sposób
   - Jedna linijka z pliku = jeden obiekt
   - Poprawne zasięgi, metody dostępowe (jeśli potrzebne), odpowiednia hierarchia to sprawy tutaj oczywiste

3. **Klasa FlightsParser**
   - Klasa zawiera metodę parseFlights(namiary na plik) zwracająca indeksowaną kolekcję obiektów z których każdy zawiera wczytane dane dotyczące jednej operacji lotniczej (przylotu lub odlotu)
     Jeżeli napotkamy niepoprawny zakres stoisk check-in np. 90-85 (gdzie jego początek jest większy od końca) to jest to sytuacja w której nie poczuwamy się do dalszego kontynuowania działania aplikacji i je przerywamy. Poprawna opcja to np. 85-90 (gdzie początek jest mniejszy od końca). Plik z błędnymi wpisami (do testów) to *odloty_bad.txt*
   - Plik loty.txt znajduje się w projekcie (można go przenieść). Dostęp do niego ma być względny i niezależny od miejsca uruchomienia aplikacji. Jest zakodowany w UTF-8.

4. **Klasa FlightsProcessor**
   - Posiada pole "allFlights" zawierające przekazaną z zewnątrz kolekcję lotów
   - Zawiera metodę getExpectedArrivals(), która z pola "allFlights" wyizolowuje i zwraca listę referencji na **przyloty** samolotów, które jeszcze nie wylądowały
   - Zawiera metodę getFlightsOrdered(), która zwraca loty z pola "allFlights" posortowane rosnąco po długości zapisu "id lotu". Kolekcja allFlights pozostaje nieposortowana.