PROI 22L - projekt

System automatycznej rejestracji pasażerów samolotu.

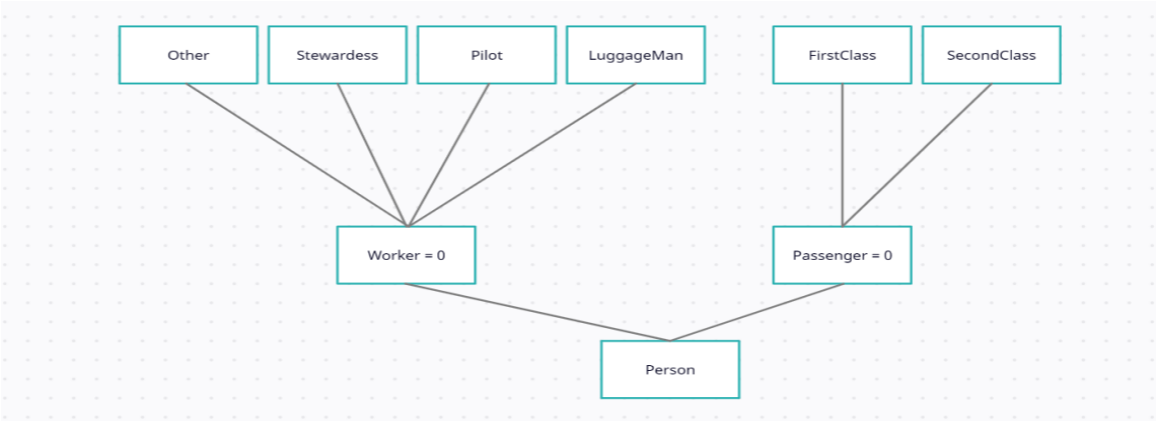
Filip Browarny

Krzysztof Kluczyński

Klasy zaimplementowane w zadaniu:

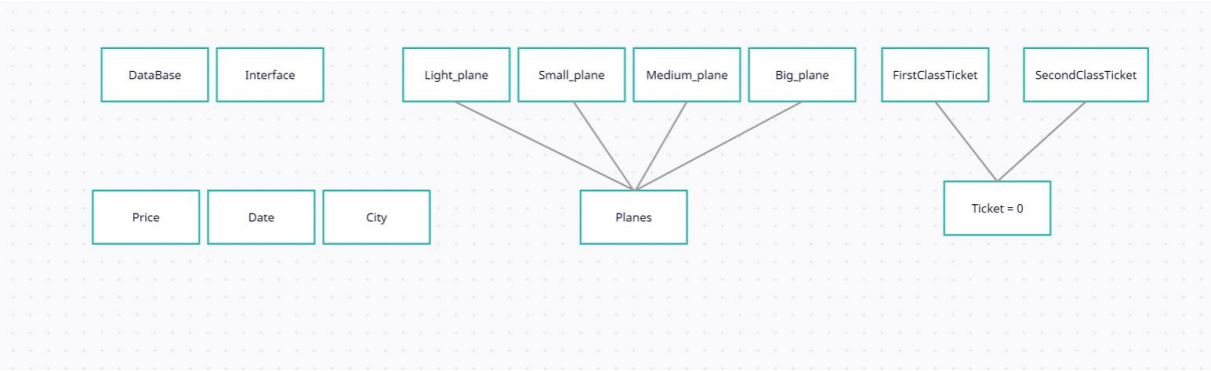
1. City - prosta klasa reprezentująca miasta pomiędzy, którymi odbywają się loty.
2. Date - data używana w wielu miejscach w programie.
3. Price - cena używana w wielu miejscach w programie.
4. Person - klasa reprezentująca pojedynczego człowieka, każdy obiekt tej klasy ma imię, nazwisko oraz pesel. Najważniejszym atrybutem jest pesel unikalny dla każdego człowieka, dzięki temu można łatwo odnosić się do konkretnego obiektu. Person jest również klasą bazową wielu innych klas opisanych poniżej. Hierarchię klas związanych z człowiekiem można zobaczyć na poniższych diagramach.
5. Passengers - w zasadzie pod tym pojęciem opiszemy całą grupę klas dziedziczących po Person, dużą rolę odgrywa w niej polimorfizm. Bazową klasą abstrakcyjną jest Passenger(dziedzicząca po person). Jej subklasy to odpowiednio FirstClassPassenger oraz SecondClassPassenger. Następuje tutaj podział względem miejsca w samolocie, FirstClassPassenger posiada jako atrybut FirstClassTicket, a SecondClassPassenger SecondClassTicket, co odpowiada przyporządkowaniu biletów w świecie rzeczywistym. Bilet posiada informację o najważniejszych parametrach lotu, którym leci dany pasażer.
6. Workers - również grupa klas dziedziczących po Person. Wygląda podobnie do pasażerów, klasa abstrakcyjna Worker zawiara subklasy: Stewardess, Pilot, LuggageMan, Other. Każda z nich różni się ceną oraz ilością obsługiwanych pasażerów.

Diagram klas z klasą bazową Person:

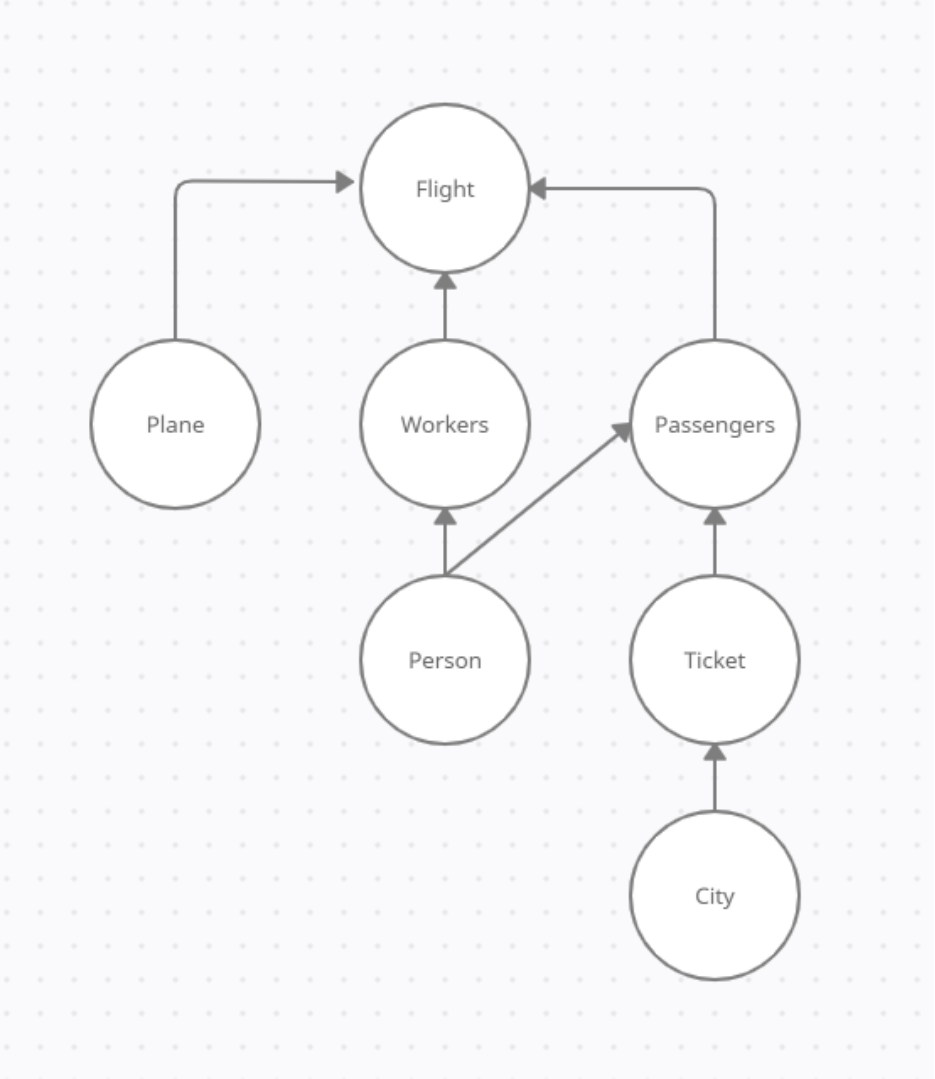


7. Ticket - w tym przypadku mówimy o 3 klasach, klasa abstrakcyjna Ticket ma subklasy FirstClassTicket i SecondClassTicket. Różnią się one sposobem przeliczania ceny za bilet
8. Planes – klasa reprezentująca samolot. Planes jest klasą wirtualną, dziedziczą po niej 4 inne klasy reprezentujące cztery rodzaje samolotów (Light_plane, Small_plane, Medium_plane, Big_plane). Klasy dziedziczące różnią się pojemnością. Klasa posiada odpowiednie gettery oraz setery.
9. Flight – klasa reprezentująca lot. Posiada pola: id, datę, miejsce wylotu, miejsce lądowania, samolot który ją obsługuje przechowywane w postaci inteligentnego wskaźnika na klasę abstrakcyjną Planes, wektory przechowujące pracowników, wektory pasażerów pierwszej oraz drugiej klasy. Klasa posiada gettery, setery, metody pozwalające na dodanie nowych pracowników oraz pasażerów z zabezpieczeniem przed dodaniem tego samego peselu dwa razy. W klasie znajdują się także metody pozwalające na usuwanie pasażerów po numerze pesel, sprawdzenie czy dany pasażer ma już bilet, sprawdzenie ilości pracowników oraz pasażerów, oraz sprawdzenie czy lot ma jeszcze wolne miejsca.
10. Data_base – klasa odpowiadająca za pobranie danych z plików txt oraz wygenerowanie lotów, posiada różne kolekcje obiektów potrzebnych w symulacji, w większości został tutaj użyty vector, wynika to z łatwości korzystania z niego oraz z szerokiej gamy dostępnych na nim operacji. Najważniejszy jest tutaj vector lotów oraz vector wszystkich pasażerów, którzy zostali wylosowani w danej symulacji(jest to wektor shared_ptr na klasę abstrakcyjną Passenger). Dostępne metody:
    a. Import_passengers() – odpowiada za stworzenie wektora pasażerów. Pasażerowie na tym etapie nie są przyporządkowani do żadnego lotu, mają imię nazwisko oraz pesel. Metoda wywołuje się automatycznie w konstruktorze.
    b. Import_workers() – odpowiada za stworzenie wektorów z pracownikami. Pracownik jest przyporządkowany do konkretnego lotu. Ta metoda również jest wywoływana w kontruktorze.
    c. Import_flights() – odpowiada za stworzenie lotów wraz z samolotem oraz przyporządkowanie im pracowników. Na tym etapie loty nie mają jeszcze żadnych pasażerów. Metoda wywoływana jest w konstruktorze.
    d. assignPassengers() – metoda odpowiadająca za symulacje. Przyporządkowuje losową ilość losowych pasażerów do lotów. Ilość pasażerów może być od 2 do ilości miejsc w samolocie. Klasa wywoływana jest w konstruktorze.
11. Exceptions – wszystkie własne wyjątki zdefiniowane są w tym pliku.
12. Interface - klasa zajmująca się obsługą interfejsu i przeprowadzeniem symulacji.

Klasy, które zostały pominięte w poprzednim diagramie:



Lot, czyli główna klasa w zadaniu, w uproszczeniu:



Funkcja Geo distance – funkcja znajdująca się w pliku ticket.cpp. Jako argumenty przyjmuje dwa stringi będące nazwami miast. Jej wynikiem jest przybliżona odległość między tymi

miastami, obliczona na podstawie ich współrzędnych geograficznych. Na podstawie odległości liczona jest bazowa cena biletu. Ta cena jest mnożona razy odpowiedni mnożnik w metodzie getRealPrice() klas pochodnych od klasy Ticket, w ten sposób różne bilety mają różną cenę.

Pliki txt:

1. Workers.txt – zawiera pracowników obsługujących loty w formacie:
   Imię Nazwisko Pesel id_lotu rola(1-stewardesa, 2-pilot, 3-bagażowy, 4-inna rola)
2. People.txt – zawiera listę osób spośród, których są losowani pasażerowie do konkretnych lotów, zagwarantowaliśmy unikalnośc peseli.
   Format:
   Imię Nazwisko Pesel
3. Flights.txt – zawiera wszystkie loty w przyjętym przez nas przedziale czasowym.
   Format:
   Id_lotu dzień miesiąc rok miejsce_wylotu miejsce_lądowania nazwa_samolotu typ_samolotu

Użytkownik może podać własne ścieżki do plików przy wywołaniu programu w kolejności odpowiednio people, workers, flights. Jeśli jednak tego nie zrobi program wykona się z domyślnymi ścieżkami do plików, gwarantuje to poprawność danych zawartych w tych plikach.

Opis działania symulacji:

Symulacja polega przeprowadzaniu lotów pomiędzy kilkoma miastami w okresie czasowym od 01.05.2020 do maksymalnie 09.05.2020. Użytkownik może ją jednak skrócić, przy rozpoczęciu programu jest pytany o to jak długo powinna trwać symulacja. Następnie do każdego lotu jest przypisywana losowa liczba pasażerów w zakresie jego maksymalnej pojemności. Pasażerowie również są losowani z puli 1000 osób znajdujących się w pliku people.txt. Każda takie przypisanie pasażera skutkuje wyświetleniem w konsoli komunikatu o zakupie biletu. Komunikaty te są wypisane w dzień danego lotu. Dodatkowo określona liczba pasażerów może zmienić swoje miejsca z pierwszej klasy na drugą. Pod koniec dnia jest wyświetlany komunikat z podsumowaniem. Informuje on użytkownika o tym jakie loty odbyły się danego dnia, jaka była liczba miejsc zajętych(wartość inna w każdej symulacji) oraz wypisywana jest obsługa danego lotu. Potem następuje przejście do następnego dnia i program zapętla się aż do przejścia przez liczbę dni jaką wybrał użytkownik.

Opis sytuacji wyjątkowych i ich obsługi:

Wszystkie wyjątki są zdefiniowane i exceptions.h i zaimplementowane w exceptions.cpp, obsługują one wprowadzenie złych danych przez użytkownika zarówno na wejściu programu(podawanie plików txt przy wywołaniu oraz odczytanie z nich danych) jak i podczas samego wykonania(wybór długości symulacji). Zabezpieczają też program na poziomie niższych klas, np. w klasach daty i ceny blokują możliwość ustawienia nierealnych parametrów, w klasie flight nie pozwalają na dodanie pasażera, który spowoduje ,,przepełnienie samolotu" ,lub też, którego pesel się powtarza. Używane są też w innych sytuacjach wyjątkowych, które powodują nierealne/niedeterministyczne działanie programu.

Podział obowiązków:

Na początku wspólnie przeanalizowaliśmy problem, rozpisaliśmy potrzebne nam klasy i podzieliliśmy się pracą. Każdy z nas dostał zadanie do wykonania w określonym czasie, następnie w miarę możliwości wspólnie łączyliśmy kod w całość, dzieląc mniejsze zadania między sobą. Dokładniejszy podział oraz pracy oraz terminy pokazują commity w repozytorium, które od prawie miesiąca było systematycznie uzupełniane.