Projekt Apteka
Programowanie Obiektowe
Semestr 23L

Michał Pióro
Damian Baraniak

Maj 2023

Spis treści

1 Założenia projektu

1. Stan początkowy symulacji jest opisany w pliku konfiguracyjnym JSON, do którego ścieżka jest podawana jako argument wywołania
2. W ciągu jednostki czasu farmaceuta jest w stanie wydać do 3 leków
3. Każdy klient ma współczynnik prawdopodobieństwa chęci zamiany leków i przy każdym leku na jego podstawie jest losowane czy szukamy zamiennika.
4. Jeśli zamienniki będą szukane to długość całej transakcji jest wydłużona o 1 jednostkę czasu.
5. Poszukiwanie zamienników odbywa się na początku transakcji.
6. Pod koniec jednostki czasu losowane jest czy najdłużej pracujący farmaceuta który jest wolny idzie na przerwę i zamyka okienko, o ile pozostanie jedno otwarte okienko. Priorytet ma okienko o niższym indeksie
7. Jeżeli w aptece jest zamknięte okienko ma ono szansę zostać otworzone
8. Na początku jednostki czasu losowane ilu klientów przyjdzie do apteki.
9. Jeden lek może być zmieniony jedynie na inny jeden lek, zamiana 1→2 czy 2→1 jest zakazana
10. Przy zamianie leki są porównywane poprzez cenę i działanie
11. To czy lek jest na receptę ustalane zostaje na podstawie zawartej ilości substancji aktywnej oraz jej rodzaju. Dla każdego typu leku te wartości są różne.
12. Prawdopodobieństwo zamknięcia/ okienka jest uzależnione od ilości okienek w aptece

2 Przebieg symulacji

2.1 Inicjalicacja symulacji oraz dane wejściowe

Wywołując plik wykonawczy, należy podać argument –path będący ścieżką do pliku JSON z parametrami opisującymi stan początkowy symulacji. W podanym pliku muszą znaleźć się takie informacje:

- nTurns - liczba iteracji jaka powinna się wykonać.
- nCounters - maksymalna liczba okienek znajdujących się w aptece.
- nOpenedCounters - początkowa liczba otwartych okienek w aptece
- nStartingClients - liczba klientów znajdująca się w aptece w momencie rozpoczęcia symulacji
- nMedicinesToCreate - liczba różnych leków jakie mają zostać wygenerowane na czas trwania symulacji.
- pathToFirstNameList - ścieżka do pliku tekstowego zawierającego listę imion dla klientów.
- pathToLastNameList - ścieżka do pliku tekstowego zawierającego listę nazwisk dla klientów.
- pathToMedicineNameList - ścieżka do pliku tekstowego zawierającego listę z nazwami leków.
- outputPath - ścieżka prowadząca do docelowej lokalizacji pliku zapisującego stan aplikacj.

Pierwsze pięć wartości, których klucze zaczynają się od małego n, powinny być dodatnimi liczbami naturalnymi. Ścieżki do plików powinny istnieć, należy zwrócić uwagę na względność ścieżek. Jako outputPath, akceptowana jest ścieżka do samego folderu, kończąca się na /, albo ścieżka kończąca się nazwą pliku .log (np. /Desktop/output.log). Na podstawie podanych danych powstanie symulacja generując odpowiednią liczbę losowych klientów oraz leków.

2.2 Przebieg iteracji

2.2.1 Przybycie nowych klientów

Przy wykorzystaniu rozkładu geometrycznego generowana jest ilość nowo przybyłych klientów. Następnie na podstawie plików z możliwymi imionami oraz nazwisakami, a także leków dostępnych w aptece generowani są w sposób pseudolosowy klienci wraz ze wszystkimi swoimi parametrami i są dodawani do kolejki. Czas trwania obsługi klienta jest ustalany w zależności od ilości leków na liście (maks 3 leki na iterację). Dodatkowo w przypadku szukania zamiennika dla pojedynczego leku dodawana jest 1 iteracja oraz w przypadku braku wystarczającej ilości leku w aptece również dodawana jest 1 iteracja. W przypadku braku leku w aptece zmniejszana jest jego ilość na liście klienta lub w przypadku całkowitego braku, jest on całkowicie z tej listy usuwany.

2.2.2 Przypisywanie klientów do wolnych okienek

W tym kroku klienci są pobierani z kolejki oraz są przydzielani do wolnych otwartych okienek przy wykorzystaniu klasy Transaction.

2.2.3 Dekremnetacja transakcji

Liczniki pozostałego czasu trwania wszystkich transakcji są dekrementowane, tak aby wiedzieć kiedy można zakończyć symulację.

2.2.4 Zwalnianie okienek

Usuwane są z listy obecnych te transakcje, które się w danej iteracji zakończyły, zwalniają one okienka.

2.2.5 Zarządzanie otwarciem okienek

W sposób losowy wybierana jest akcja zamknięcia lub otwarcia okienka i następnie losowo następuje decyzja czy się ona odbędzie. Jeśli tak to otwierane jest losowe okienko lub zamykane najdłużej pracujące.

2.3 Zakończenie symulacji

Normalne zakończenie symulacji odbywa się po odbyciu zadanej liczby iteracji. W tym przypadku kończone są transakcje klientów wcześniej niż powinny (okienka chwilowo szybciej obsługują klientów). Wyświetlane jest krótkie podsumowanie stanu apteki.

3 Testowanie

3.1 Przeprowadzone testy

Dla każdej z klas reprezentujących realne części apteki, zostały stworzone testy jednostkowe, które sprawdzają poprawność działania wykorzystywanych funkcjonalności klas. Zostały one także stworzone dla kalsy Transaction, w krórej dało się ominąć fakt występowania losowości. Natomiast dla pozostałych klas, z powodu losowości, zostały przeprowadzone wspólnie testy z wykorzystaniem debbuger-a, sprawdzające pełną funkcjonalność symulacji.

3.2 Sytuacje wyjątkowe

3.2.1 Brak klientów do obsługi

W momencie gdy kolejka jest pusta oraz nie jest przeprowadzana żadna transakcja, co znaczy że nie ma klientów w aptece, symulacja zostaje przerwana.

3.2.2 Nieprawidłowe dane wejściowe

Przed rozpoczęciem symulacji, w trakcie pobierania danych z pliku JSON, jest weryfikowana ich poprawność i zgodność z logiką, w momencie błędów program się przerywa i wyświetla odpowiedni komunikat.

4 Dane wyjściowe programu

W wyniku działań apteki powstają różne komunikaty inforumjące o zmianie wewnętrznego stanu, są one wyświetlane w konsoli oraz zapisywane do pliku logującego, którego ścieżka została podana jako outputPath. Każdy komunikat zaczyna się od podania aktualnego numeru kolejki oraz tagu precyzującego do jakiego typu zdarzenia doszło.

5 Opis klas

5.1 Medicine

Jest to abstrakcyjna klasa bazowa reprezentująca lek. Przechowuje ona w sobie dane takie jak: nazwa leku, substancja aktywna, ilość substancji aktywnej w dawce, leczona dolegliwość, cena, oraz ilość w aptece. Na podstawie substancji aktywnej oraz jej ilości określane jest to czy dany lek jest na receptę. Te paramtery są różne w zależności od typu leku. Cena leku jest ceną hurtową, więc dla każdego typu leku doliczana jest różna prowizja apteki.

Klasy pochodne Medicine - Typy leków

5.1.1 Capsules

Klasa reprezentująca kapsułki. Dodatkowym jej polem jest ilość kapsułek w opakowaniu.

5.1.2 Drops

Klasa reprezentująca krople. Dodatkowym jej polem jest objętość w mililitrach.

5.1.3 Ointment

Klasa reprezentująca maść. Dodatkowym jej polem jest objętość w mililitrach.

5.1.4 Powders

Klasa reprezentująca proszki. Dodatkowym jej polem jest ilość saszetek w opakowaniu.

5.1.5 Syrup

Klasa reprezentująca syrop. Dodatkowym jej polem jest objętość w mililitrach.

5.1.6 Tablets

Klasa reprezentująca tabletki. Dodatkowym jej polem jest ilość tabletek w opakowaniu.

5.2 Price

Klasa reprezentująca cenę w złotówkach z dokładnością do groszy. Posiada przeciążone operatory działań oraz porównywania niezbędne do jej wykorzystania.

5.3 ShoppingItem

Jest to klasa przechowująca wskaźnik do leku znajdującego się w inwentarzu apteki oraz ilość którą chce zakupić dany klient. Służy do ułatwienia przechowywania danych w liście z lekami, które chce zakupić dany klient.

5.4 ShoppingList

Klasa przechowująca obiekty typu ShoppingItem. Służy do przechowywania listy leków, które chce zakupić dany klient. Kolekcją wykorzystaną do przechowywania danych w tym przypadku jest **lista**. Została wybrana ze względu na to że w liście pojawiają się czeste modyfikacje typu dodawanie nieznanej wcześniej ilości leków oraz usuwanie ich z listy w momencie, gdy brakuje ich w aptece, a lista przyspiesza takie procesy.

5.5 Client

Jest to abstrakcyjna klasa reprezentująca klienta. Przechowuje w sobie imię, nazwisko, listę leków do zakupu oraz prawdopodobieństwo podjęcia przez klienta akcji takich jak zamiana leku na tańszy. Klasy pochodne odpowiadają za rozróżnienie klientów, którzy chcą rozliczyć się na paragon lub fakturę.

5.5.1 BusinessClient

Reprezentuje klienta robiącego zakupy na fakturę. Podatek VAT wynosi dla niego 8%.

5.5.2 IndividualClient

Reprezentuje klienta robiącego zakupy na paragon. Podatek VAT wynosi dla niego 23%.

5.6 ClientsQueue

Reprezentuje kolejkę klientów znajdującą się w aptece. Do przechowywania klientów została wykorzystana kolekcja typu **queue**, ponieważ umożliwia ona zastosowanie systemu zarządzania danymi FIFO. W ten sposób klienci są obsługiwani w takiej samej kolejności, w której przybywają do apteki, czyli tak jak to jest w rzeczywistości.

5.7 Counter

Klasa opisująca okienko w aptece. Każda instancja ma swoje niepowtarzalne id, okienko może być otwarte/zamknięte oraz wolne/zajęte. Dodatkowo posiada przeciążony operator imkrementacji pozwalający w łatwy sposób zwiększąć długość pracy każdego okienka. Na końcu każdej kolejki wolne okienko, które pracuje najdłużej może zostać zamknięte, a inne wcześniej zamknięte okienko może zostać otwarte.

5.8 CountersList

Klasa pozwalająca składować odpowiednią ilość okienek. Okienka są składowane jako unikalne wskaźniki wewnątrz zbioru. Nie jest przewidziane dodawanie nowych okienek poza konstruktorem, a jego argumentami jest ogólna liczba okienek w aptece i liczba otwartch na początku okienek. Do poszczególnych okienek można dostawać sie za pomocą ich identyfikatora.

5.9 Inventory

Klasa składująca wszystkie istniejące leki w dostępnym ekwipunku. Główną funkcjonalnością jet mapa dzieląca różne leki na podstawie ich działania, leki o takim samym działaniu są składowane wewnątrz zbioru, jako shared_pointer. Taka konstrukcja pozwala w łatwy dostęp do odpowiednich leków, oraz losowanie odpowiednich leków przy tworzeniu listy zakupów.

5.10 Pharmacy

Reprezentuje obiekt, którym jest apteka. Przechowuje w sobie obiekty typu: Inventory, ClientsQueue, CountersList. Pozwala na korzystanie z ich metod potrzebnych w symulacji.

5.11 Transaction

Klasa reprezentująca transakcję danego klienta. Przechowuje w sobie wskaźnik na klienta, referencję do magazynu apteki oraz referencję do przypisanego klientowi okienka. Odpowiada za pobranie leków z apteki, zarządzanie listą klienta w przypadku gdy brakuje leków w aptece, zajęcie i zwolnienie okienka oraz szukanie zamienników, gdy jest taka potrzeba. Transakcja przechowuje też czas jej trwania, tak aby, gdy zakończy się jej czas trwania, zostało zwolnione okienko.

5.12 FileManager

Klasa odpowiadająca za obsługę różnych plików pojawiajacych się w programie. Posiada jedynie metody statyczne, główną funckjonalnością jest konstrukcja symulacji. Również wszystkie dane są weryfikowane pod kątem poprawności. Dodatkowo pozwala stworzyć dane pomocnicze jak listy dostępnych imion, nazwisk, oraz nazw leków.

5.13 Logger

Klasa pozwalająca wyświetlać komunikaty w konsoli i do zapisu komunikatów do odpowiedniego pliku. Posiada przeciążony operator « dzięki któremu może przyjmować komunikaty. Dodatkowo działa funkcjia std::endl;

5.14 Simulation

Klasa odpowiedzialna za przeprowadzenie symulacji. Przechowuje aptekę, której to działanie jest symulowane, czas trwania symulacji, dane potrzebne do generowania imion oraz listę obecnie trwających transakcji. Z kolekcji została wybrana lista, ze względu na częstą zmianę długości. Pozwala na tworzenie w sposób pseudolosowy różnych danych potrzebnych do stworzenia oraz korzystania z apteki.

6 Podział zadań

Michał	**Damian**
Client	Counter
ClientsQueue	CountersList
Medicine	Inventory
Transaction	Logger
ShoppingList	FileManager
Simulation	
Pharmacy	

7 Wykorzystane biblioteki

- Catch2 - Wykorzystana przy testach jednostkowych wewnątrz projektu.
- json - Wykorzystana przy czytaniu danych z podanego pliku konfiguracyjnego.
- cxxopts - Wykorzystywana przy obsłudze argumentów wywołania programu z poziomu terminala.