Symulator sklepu budowlanego

Jakub Ostrzołek (@jostrzol)
Jakub Marcowski (@jmarcows)

Spis treści

Spis treści **2**

Założenia projektowe **3**

Hierarcha klas **3**

Biblioteka statyczna - struktura sklepu 3

Program wykonywalny - symulator 4

Przeprowadzenie symulacji **6**

Wykorzystane elementy biblioteki STL **6**

Kontenery 6

Czas 6

Liczby losowe 6

Strumienie 7

Inne 7

Obsługa sytuacji wyjątkowych **7**

Założenia projektowe

Przedmiotem projektu jest symulacja sklepu budowlanego.
Sklep ma swoich pracowników, klientów i kasy. Uczestnicy symulacji wykonują dozwolone dla nich akcje przez cały czas jej działania.
Czas symulacji jest podzielony na tury, z czego każda zajmuje 15 minut czasu wirtualnego i 5 sekund czasu rzeczywistego. Efekty symulacji można obserwować w terminalu oraz w wygenerowanym pliku log.txt po jej zakończeniu.
Szczegóły symulacji wczytywane są poprzez pliki konfiguracyjne w katalogu z plikiem wykonywalnym:

- names.txt - plik z imionami, które będą wykorzystywane podczas symulacji do generowania osób
- addresses.txt - plik z adresami, które będą wykorzystywane podczas symulacji do generowania obiektów
- items.csv - plik z artykułami sklepowymi, które będą dostępne do kupienia w sklepie.

Oprócz tego program przyjmuje 2 parametry z konsoli - ilość kas sklepowych oraz ilość pracowników w sklepie oraz przed wystartowaniem symulacji pyta użytkownika o ilość tur, jaka ma się rozegrać.

Hierarcha klas

Program został podzielony na dwie główne części - bibliotekę obsługującą strukturę sklepu oraz część wykonawczą - symulator.

Biblioteka statyczna - struktura sklepu

Wszystkie klasy należące do biblioteki zostały przedstawione w postaci uproszczonego diagramu UML na następnej stronie.
Krótki opis ważniejszych klas z biblioteki:

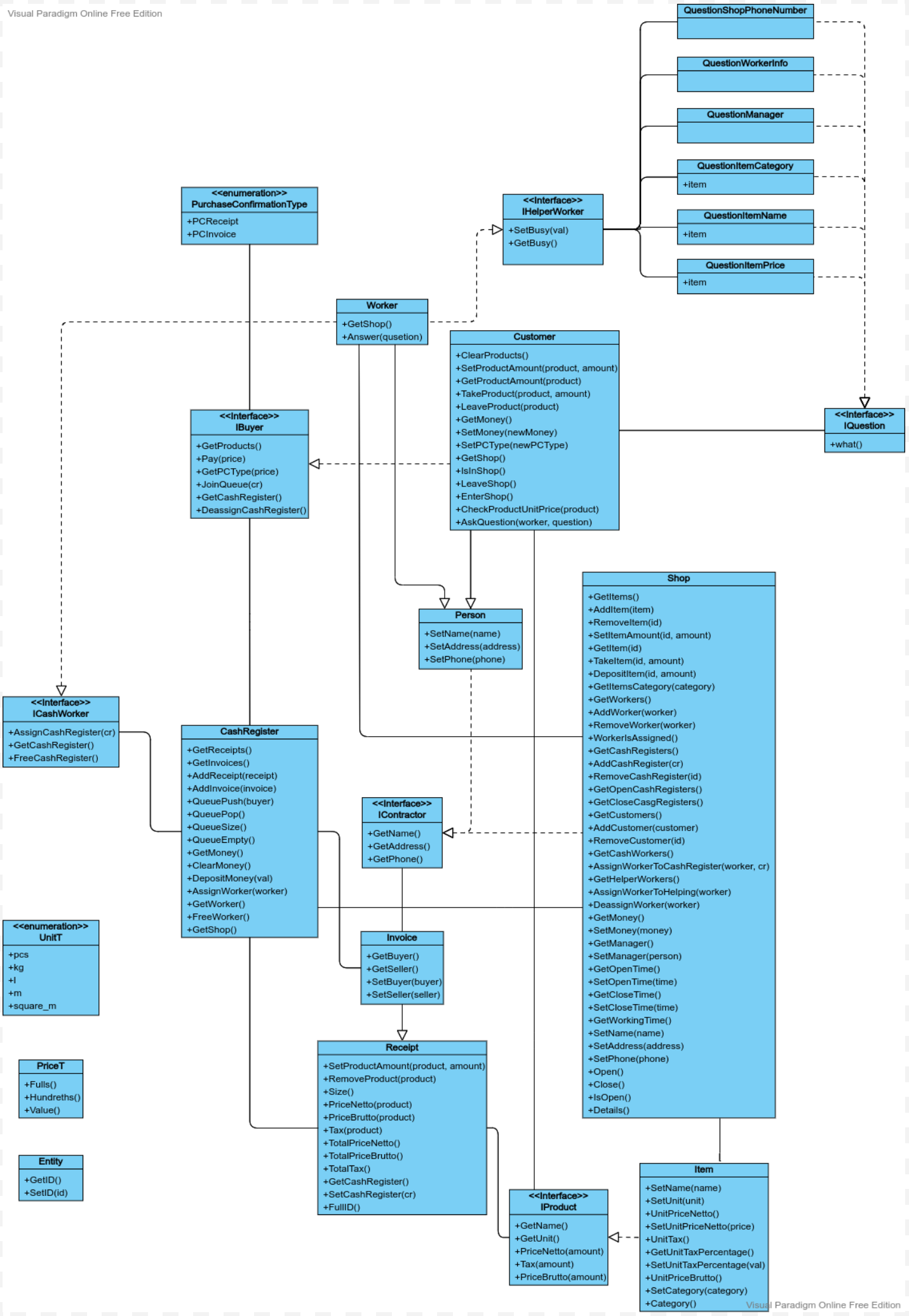
- Shop - realizuje IContractor. Największa klasa będąca właścicielem prawie wszystkich pozostałych klas składowych. Posiada głównie funkcje dostępowe do obiektów składowych oraz kilka funkcjonalności dla siebie specyficznych, takich jak czas otwarcia sklepu czy zamknięcie/otwarcie sklepu.
- Customer - realizuje IBuyer, specjalizuje Person. Klasa reprezentująca klienta. Jej główne funkcjonalności skupiają się wokół zarządzania swoimi produktami.
- Worker - realizuje ICashWorker i IHelperWorker, specjalizuje Person. Klasa reprezentująca pracownika sklepu. Zasadniczo robi tyle, ile wymagane jest do realizacji interfejsów przez nią realizowanych.
- CashRegister - klasa reprezentująca kasę sklepową. Posiada funkcjonalności pozwalające na zarządzanie rachunkami i fakturami oraz przetrzymywanie pieniędzy.
- Person - realizuje IContractor. Klasa reprezentująca osobę, pozwala na zarządzanie jej imieniem, adresem i telefonem.
- IBuyer - interfejs komunikacji między kasą sklepową (kolejką do niej) oraz klientem. Wymaga metod dostępu do produktów nabywcy oraz możliwości zapłaty, oraz metod interakcji z kasą sklepową.
- ICashWorker - interfejs komunikacji pracownika obsługującego kasę oraz tej kasy. Wymaga metod obsługujących przypisanie pracownika do kasy.

- IHelperWorker - interfejs komunikacji pracownika i klienta z pytaniem.
- IContractor - interfejs umożliwiający identyfikację kontrahenta na fakturze.
- IQuestion - interfejs umożliwiający stawianie pytań przez klientów.
- Receipt - klasa reprezentująca rachunek. Jej funkcjonalności skupiają się na obsłudze produktów na rachunku.
- Invoice - specjalizuje Receipt. Klasa reprezentująca fakturę. Zawiera informacje o kontrahentach - zbywcy i nabywcy.
- IProduct - interfejs komunikacji między artykułami a rachunkiem/klientem. Wymaga metod dostępu do podstawowych informacji o produkcie.
- Item - realizuje IProduct. Klasa reprezentująca artykuł sklepowy. Posiada dodatkowe funkcjonalności związane z kategorią artykułu.
- Entity - klasa reprezentująca obiekt z identyfikatorem ID. Wiele klas po niej dziedziczy (nie jest to zaznaczone na diagramie, aby nie zaciemniać istotniejszych relacji)
- PriceT - klasa reprezentująca cenę/pieniądze w postaci stałopozycyjnej.

Program wykonywalny - symulator

Symulator składa się z dwóch głównych klas i istotnego szablonu:

- Simulation - klasa reprezentująca samą symulacje. Zawiera metody pozwalające na przeprowadzenie symulacji. Oprócz tego zarządza czasem wirtualnym w symulacji. Korzysta z szablonu Actions<> dla obiektów takich jak klienci i pracownicy sklepowi.
- Actions<> - szablon reprezentujący akcje dostępne do wylosowania w danej turze wraz z wagami.
- ObjectGenerator - klasa zajmująca się losowym generowaniem obiektów do sklepu dzięki informacjom zawartym w plikach konfiguracyjnych. Jest na stałe związana z danym sklepem i zapewnia niegenerowanie “duplikatów” - obiektów o identycznych parametrach.



Przeprowadzenie symulacji

Symulacja podzielona jest na 3 etapy:

1. Przygotowanie symulacji - tutaj konstruowany jest sam obiekt symulowanego sklepu oraz wszystkie obiekty wymagane do poprawnego przebiegu symulacji, takie jak pracownicy czy klienci, do których konstrukcji wykorzystywana jest klasa ObjectGenerator. Następnie tworzony jest obiekt symulacji tego sklepu, dostosowywane są parametry symulacji oraz jest ona uruchamiana.
2. Pętla symulacji - tutaj rozgrywane są wszystkie akcje wykonywane przez obiekty podlegające symulacji. Większość tych akcji jest na podstawie prostych, częściowo losowych algorytmów, udających rozumowanie ludzi. Pętla ta jest wykonywana przez zadaną przez użytkownika ilość tur.
   Po każdym zakończonym wirtualnym dniu symulacji do sklepu trafia nowa porcja towarów, a każdy z klientów dostaje trochę pieniędzy.
3. Zakończenie symulacji - teraz zapisywany jest plik wyjściowy symulacji, a wszystkie obiekty symulacji są niszczone.

Wykorzystane elementy biblioteki STL

Kontenery

- std::unordered_map - wykorzystywane bardzo szeroko w programie jako główny kontener na obiekty dziedziczące po klasie Entity, np. do przechowywania artykułów sklepowych, pracowników, klientów, itp.
- std::vector - wykorzystywane sporadycznie w porównaniu do std::unordered_map raczej jako obiekt tymczasowy niż faktyczny kontener na obiekty, ponieważ w przypadku gdy obiekty identyfikowane są przez ID łatwiej trzymać je w hashmapie; wykorzystywane np. w klasie ObjectGenerator do zwracania wskazań na obiekty wygenerowane do sklepu
- std::queue - wykorzystywane jako kolejka klientów do kasy
- std::transform, std::for_each, std::all_of - wykorzystywane do manipulowania kontenerami w wygodny sposób

Czas

- std::chrono::duration - wykorzystywane głównie w klasie Simulation do odmierzania czasu bezczynności oraz obsługi czasu wirtualnego - czas dnia
- std::chrono::system_clock - wykorzystywane do ziarna dla generatorów losowych oraz odmierzania czasu bezczynności

Liczby losowe

- std::mt19937 - jako główny generator liczb losowych
- std::uniform_int_distribution, std::normal_distributuion, std::discrete_distribution - jako rozkłady liczb losowych używane w wielu miejscach w klasie Simulator (i nie tylko)

Strumienie

- std::cout - do wypisywania do wyjścia standardowego
- std::(i/o)fstream - do wypisywania do pliku wyjściowego oraz czytania danych konfiguracyjnych
- std::stringstream - najczęściej jako tymczasowe obiekty tworzone w celu wygenerowania docelowego ciągu znaków
- manipulatory z nagłówka <iomanip> - do formatowania danych wypisywanych do strumieni w wielu miejscach w programie

Inne

- std::string - wykorzystywane szeroko w całym projekcie jako ciągi znaków reprezentujące np. nazwy, adresy, itp.
- std::hash - do tworzenia map z niestandardowymi obiektami-kluczami
- wyjątki z nagłówka <exception> - wykorzystywane jako klasy bazowe dla wyjątków zdefiniowanych w programie

Obsługa sytuacji wyjątkowych

Ze względu na wpływ na wydajność i konwencję języka C++ w projekcie zostało ograniczone użycie mechanizmu wyjątków.

Jeżeli funkcja może się nie powieść, a zwraca wskazanie na jakiś obiekt, to zamiast rzucania wyjątku funkcja zwraca nullptr, np. CashRegister::QueuePop. Zastosowanie takiego podejścia powoduje, że w programie w wielu miejscach wskazania są przyrównywane do nullptr, np. Receipt::FullID, Shop::DeassignWorker.

Jeżeli funkcja może się nie powieść, czasem zwraca swój status, aby wywołujący mógł zdecydować o dalszych krokach, np. IBuyer::Pay, Shop::TakeItem.

Jeżeli funkcja przyjmuje strumień jako argument, przy nieudanym odczycie/zapisie, w celu sygnalizacji błędu ustawiany jest failbit, jak np. w operatorze >> typu UnitT.

W reszcie przypadków zastosowany został standardowy system wyjątków, np. w operatorach klasy PriceT w razie wygenerowania ceny ujemnej, w wielu metodach klasy Shop w razie próby manipulacji obiektów nie z tego sklepu lub przy próbie przypisania zajętego już identyfikatora ID. Ten system został również wykorzystany przy interakcji z użytkownikiem - przy badaniu poprawności plików, argumentów oraz standardowego wejścia.

Dzięki użytemu podejściu udało się mocno ograniczyć użycie instrukcji try … catch, co potencjalnie przyspiesza działanie programu.