Projekt 1.4 - Symulacja działania banku

Autorstwa:

Patryk Zdziech

Wojciech Styczeń

Główne założenia:

* Symulacja generuje określoną podczas inicjalizacji liczbę klientów oraz pracowników.
* Wszyscy wygenerowani pracownicy oraz klenci biorą udział w symulacji.
* Zakładamy, że dla każdego pracownika jest dostępne wolne stanowisko.
* Kasa obsługuje naraz jednego klienta.
* Od rodzaju transakcji zależy, jakie kwalifikacje są potrzebne do jej wykonania i ile czasu ona zajmuje.
* Informacje o przebiegu symulacji są wyświetlane na konsoli oraz zapisywane w pliku tekstowym o domyślnej nazwie ‘BankLogs’.

Użyte biblioteki:

#include <iostream>

#include <string>

#include <set>

#include <vector>

#include <deque>

#include <memory>

#include <map>

#include <algorithm>

#include <algorithm>

#include <iostream>

#include <exception>

#include <iterator>

Hierarchia klas:

Klasa reprezentująca dokument tożsamościowy:

**class** **Id**

**private:**

string personalNr; // 11 digits

string type;

**public:**

**static** **const** vector<std::string> validIdTypes;

Id(string IdNr, string IdType);

Id();

**void** **setRandomNr**(); // 11 digits

**void** **setRandomType**();

string **getPersonalNr**();

string **getType**();

**void** **changePersonalNr**(string newNr);

**void** **changeType**(string newType);

**bool** **checkPersonalNr**(string n);

**bool** **checkType**(string& t);

Możliwe rodzaje dokumentu to: "Identity Card", "Passport", "Driver's License" Numer dokumentu ma 11 cyfr, na potrzeby zadania można wygenerować losowy. Wykorzystywane w klasie Client.

Klasa reprezentująca imię i nazwisko:

**class** **Name**

**private:**

string firstName;

string lastName;

**public:**

Name(string first, string last);

Name(**const** Name& n);

string **getFirstName**() **const**;

string **getLastName**() **const**;

**void** **changeFirstName**(string newFirstName);

**void** **changeLastName**(string newLastName);

**void** **checkCaps**(string& s);

Imię i nazwisko jest standaryzowane tak by zaczynało się z dużej litery i było pisane dalej małymi. Wykorzystywane w klasach Employee oraz Client.

**class** **Product**

responsibility requiredResponsibility;

**int** time;

string name;

**public:**

Product(responsibility requiredResponsibility, **int** time, string name);

**bool** **operator**==(**const** Product& p2);

responsibility getResponsibility() **const**;

**void** **setResponsibility**(responsibility requiredResponsibility);

**void** **reducetime**();

**void** **reducetime**(**int** t);

**bool** **isDone**();

**int** **getTime**() **const**;

**void** **setTime**(**int** time);

**const** string& **getName**() **const**;

**void** **setName**(**const** string &name);

Klasa reprezentująca klienta banku:

**class** **Client**

**private:**

Name name;

Id id;

**int** clientCode; // 6 digits / can't start with zero

string PIN; // 4 digits

Product product;

**public:**

**static** set<**int**> takenCodes;

Client(string firstName\_, string lastName\_, string idType\_, string idNr\_, **int** clientCode\_, string PIN\_, Product product);

Client(string firstName\_, string lastName\_, Product product\_);

**void** **setRandomClientCode**();

**void** **setRandomPIN**();

**bool** **checkIfCodeIsUnique**(**const** **int**& clientCode);

string **getFirstName**();

string **getLastName**();

string **getIdType**();

string **getIdNr**();

**bool** **reduceProductTime**();

**int** **getClientCode**() **const**;

string **getPIN**();

Product **getProduct**();

**bool** **checkIfClientCodeValid**(**const** **int** &code);

**bool** **checkIfPINValid**(**const** string &pin);

**void** **changeClientCode**(**int** newClientCode);

**bool** **comparePIN**(string PIN);

**bool** **compareClientCode**(**int** clientCode);

**const** Name& getName() **const**;

Zawiera imię i nazwisko, dokument tożsamości, osobisty kod klienta (6 cyfr), PIN bakowy (4 cyfry) i produkt. Kody klientów nie mogą się powtarzać, co jest sprawdzane. Na potrzeby symulacji możliwe jest generowanie losowego PIN’u i kodu.

Uprawnienia bankowe:

**enum** permission {analitic,bank\_assistant,cashier,director,financial\_manager,data\_specialist,account\_manager, lastPermission};

**enum** responsibility

{cash\_withdraw, transfer, fast\_transfer, deposit\_cash, retrieve\_user\_data, change\_user\_data, check\_account, credit\_offer, account\_offer, new\_user\_help, lastResponsibilty};

**extern** map<permission, set<responsibility>> possibleResponsibilities;

Zawiera możliwe operacje banku oraz uprawnienia pracowników. Skojarzone są ze sobą za pomocą mapy. Każdej posadzie odpowiada więcej niż jedno uprawnienie.

Klasa reprezentująca pracownika banku:

**class** **Employee**

Name name;

**static** set<**int**> takenNumbers;

**int** employeeNumber{}; // 4 digits

set<permission> permissions;

set<responsibility> responsibilities;

**public:**

Employee(std::string firstName, std::string lastName);

Name **getName**();

**virtual** ~Employee();

**void** **generateEmployeeNumber**();

**int** getEmployeeNumber() **const**;

**void** **setEmployeeNumber**(**int**);

string **getFirstName**();

string **getLastName**();

**const** set<permission>& getPermissionsGranted() **const**;

**void** **addPermission**(permission permissionGranted);

**void** **removePermission**(permission permissionToRemove);

**void** **clearPermissions**();

**bool** **canHaveResposibility**(**const** responsibility resp);

**bool** **addResponsibility**(responsibility responsibilityAdded);

**void** **removeResposibility**(responsibility responsibilityToRemove);

**void** **clearResponsiblities**();

**const** set<responsibility>& **getResponsibilities**() **const**;

**bool** **operator**<(**const** Employee& r);

Zawiera imię i nazwisko, unikatowy numer pracownika (4 cyfry), zestaw obowiązków wraz z uprawnieniami jakie posiada pracownik. Numer użytkownika musi być unikatowy. Na potrzeby symulacji jest generowany losowo.

Klasa reprezentująca stanowisko w banku:

**class** **Counter**

**static** set<**int**> nrsInUse;

**static** set<Employee> employeesAssigned;

**int** nr; // 4 digits

Employee employee;

deque<Client> queue;

**public:**

Counter(**int** nr\_, **const** Employee &e, deque<Client> queue\_ = {});

Counter(**const** Employee &e, deque<Client> queue\_ = {});

**void** **setRandomCounterNumber**();

**bool** **checkIfNrWithinBounds**(**const** **int** &counterNr);

**bool** **checkIfNrIsUnique**(**const** **int** &counterNr);

**bool** **checkIfEmployeeIsFree**(**const** Employee& e) **const**;

**bool** **canHandleProdukt**(Product) **const**;

**bool** **handleClient**();

**int** getNr() **const**;

Employee **getEmployee**();

deque<Client> **getQueue**();

**int** **getNrsInUseSize**();

**int** **getQueueSize**() **const**;

Client **getfront**();

**void** **changeCounterNr**(**int** newNr);

**void** **newClient**(Client newCl);

**void** **clientLeftAfterService**();

**void** **clientLeft**(Client cl);

**void** **clearQueue**();

**void** **assignEmployee**(Employee e);

**void** **dismissEmployee**(**const** Employee &e) **const**;

Zawiera przypisanego pracownika i kolejkę klientów. Każde stanowisko ma też unikatowy numer (4 cyfry) który może być wygenerowany losowo.

**canHandleProdukt -** sprawdzającą czy kient może być obsłużony przy danej kasie **handleClient –** obsługuje klienta, oznacza to że następuję postęp o jedną jednostkę czasu usługi której oczekuje klient.

**clientLeftAfterService –** obsłużony klient opuszcza kolejkę

**clientLeft –** klient wyszedł z kolejki

Klasa reprezentująca bank:

**class** **Bank**

vector<Counter> counters{};

set<Employee> employees{};

set<Client> clients{};

**static** set<string> randomFirstNames;

**static** set<string> randomSecondNames;

**public:**

Bank(**int** m, **int** n);

string **getRandomFirstName**();

string **getRandomLastName**();

**void** **loadEmployees**(**int** m);

**void** **loadClients**(**int** n);

**void** **loadCounters**();

set<Client> **loadQueues**();

**void** **loadClientToQueue**(Client c);

**void** **simulate**(**double** update\_time);

**bool** **allQueuesEmpty**();

**const** set<Client>& **getClients**() **const**;

**void** **setClients**(**const** set<Client> &clients);

**const** vector<Counter>& **getCounters**() **const**;

**void** **setCounters**(**const** vector<Counter> &counters);

**const** set<Employee>& **getEmployees**() **const**;

**void** **setEmployees**(**const** set<Employee> &employees);

**void** **loadRandomNames**(string, string);

Główna klasa służąca do przeprowadzenia symulacji. Zawiera Obecnych klientów i pracowników oraz kasy bankowe. Przy konstrukcji obiektu generowani są losowi pracownicy i klienci.

**loadClients, loadEmployees –** generuje pracowników/klientów.

**getRandomFirstName**, **getRandomLastName –** losuje imię/nazwisko z plików tekstowych: firstname.txt / surname.txt

**loadCounters –** przyporządkowuje pracowników do stanowisk.

**loadQueues -** przyporządkowuje klientów do stanowis. Klienci przypisywani są po kolei szukając odpowiednich kas mogących zrealizować ich usługi poczynając od tych gdzie kolejka jest najkrótsza.

Zaimplementowane wyjątki:

**class** **InvalidIdType**, **ClientCannotBeAsigned**, **InvalidIdNr**,

**InvalidClientCode**, **DuplicateProduct**, **ProductNotFound**, **InvalidCounterNr**, **DuplicateCounterNr**, **CounterNrNotFound**, **ClientNotFound**, **DuplicateClientCode**, **EmployeeAlreadyAssigned**, **FileNotFoundError**, **AllNumbersTakenError**, **EmployeeNumberTaken**

W zdecydowanej większości są one wywoływane w przypadku próby wprowadzenia niepoprawnych danych lub wykonania operacji, które nie są w danym momencie możliwe.

Np.:

- wyjątek **InvalidClientCode** zostaje wywołany przy próbie zainicjalizowanie obiektu klasy Client o nieporawnym kodzie.

- wyjątek **FileNotFoundError** zostaje wywołany jeśli otwarcie danego pliku się nie powiodło

- wyjątek **ClientNotFound** może zostać wywołany np. przy próbie usunięcia z kolejki klienta, którego tam w rzeczywistości nie ma.

Itd.

Symulacja:

Symulacja jest odpalana poprzez funkcję runSimulation() – przyjmującą na wejście liczbę klientów oraz pracowników oraz, ewentualnie, nazwę pliku. Wszystkie dane z symulacji wyświetlane są zarówno w konsoli jak i zapisywane w pliku tekstowym – domyślnie o nazwie ‘BankLogs.txt’.