|  |  |
| --- | --- |
|  | **AKADEMIA GÓRNICZO-HUTNICZA** |

Dokumentacja do projektu

**System bankowy**

z przedmiotu

**Języki programowania obiektowego**

Elektronika i Telekomunikacja PL, III rok

*Szymon Stolarski*

*Michał Szczepaniak-Krupowski*

Poniedziałek 16.15

prowadzący: dr inż. Rafał Frączek

17.01.2020

# Opis projektu

Stworzony przez nas projekt zawiera podstawy systemu bankowego. Zarządzają nim Administratorzy, a dostęp do systemu mają pracownicy banku, jego klienci, oraz w przypadku jednego rodzaju konta i paru funkcji globalnych – także zewnętrzni użytkownicy. System pozwala na zakładanie, usuwanie, modyfikowanie kont oraz operacje pomiędzy nimi takie jak: wysyłanie wiadomości, przelewanie środków, czy w przypadku kont o wielu właścicielach, podejmowanie wspólnych decyzji na drodze głosowań.

# Project description

The project we created contains the basics of the banking system. It is managed by Administrators, and the system is accessed by bank employees, its customers, and external users for one type of account and some global functions. The system allows you to set up, delete, modify accounts and operations between them, such as sending messages, transferring funds, or in the case of multi-owner accounts, making joint decisions by voting.

# Instrukcja użytkownika

Obsługa programu jest nieskomplikowana, wystarczy podążać za wyświetlanymi w konsoli instrukcjami, które opisują każdy podejmowany przez użytkownika krok. Trzeba mieć na uwadze fakt, że powrót do wcześniejszego menu nie jest możliwy, chyba, że samo menu informuje o takiej opcji.

# Kompilacja

Aby skompilować projekt używaliśmy domyślnego kompilatora w programie Visual Studio 2019, więc potencjalnym użytkownikom radzimy używać tego samego.

# Pliki źródłowe

Projekt składa się z następujących plików źródłowych:

*bank.h* – deklaracja wszystkich klas użytych w projekcie, ich metod oraz funkcji z nimi powiązanych

*Konto.cpp, Klient.cpp, Firma.cpp, Fundacja.cpp, Admin.cpp, Pracownik.cpp* – implementacja klas o tych nazwach co plik, ich metod oraz funkcji z nimi związanych

*communication.h, communication.cpp* – deklaracja oraz implementacja funkcji związanych ze wspólnym podejmowaniem decyzji wśród wspólników w klasach Firma oraz Fundacja

*interfejsy\_kont.h, interfejsy\_kont.cpp* – deklaracja oraz implementacja funkcji odpowiedzialnych za poruszanie się klienta po swoim koncie

*external.h, external.cpp* – deklaracja i implementacja funkcji niezależnych od reszty systemu tj. kalkulatory rat kredytowych oraz zysku z lokat

*add\_notification.cpp* – implementacja funkcji odpowiedzialnych za przyznawanie zadań pracownikom

*wykonanie\_operacji.cpp* – implementacja funkcji związanych z działaniami pracowników banku

# Zależności

brak

# Opis klas

W projekcie utworzono następujące klasy:

1. *Konto* – główna klasa w projekcie, reprezentuje Konto w systemie bankowym
2. *string get\_ID(void)* – zwraca numer identyfikacyjny konta
3. *string get\_password(void)* – zwraca hasło dostępu do konta
4. *void set\_ID(const string& ID)* – ustawia nowe ID do konta
5. *void set\_password(const string& password)* - ustawia nowe hasło do konta
6. *bool logowanie(void)* – zwraca true, gdy warunki logowania (zgodnosc wpisanych loginu I hasła) zostały spełnione
7. *Klient* – klasa pochodna klasy *Konto* – konta przeznaczone dla statystyczego użytkownika indywidualnego
8. *string get\_imie(void)* – zwraca imie właściciela
9. *string get\_nazwisko(void)* – zwraca nazwisko właściciela
10. *long double get\_saldo(void)* – zwraca stan konta
11. *long double get\_prog(void)* – zwraca aktualny próg transakcji nieautoryzowanych
12. *string get\_message(void)* – zwraca wiadomosc oczekujaca na użytkownika
13. *void set\_imie(string imie);* - ustawia imię
14. *void set\_nazwisko(string nazwisko);* - ustawia nazwisko
15. *void set\_saldo(long double saldo);* - zmienia stan konta (po wpłacie lub wypłacie)
16. *void set\_prog(long double prog);* - zmienia próg transakcji nieautoryzowanych
17. *void set\_message(string message);* - ta metoda służy do wpisywania wiadomości do odczytania przez klienta gdy sie zaloguje
18. *!!!!! bool logowanie(void);* - logowanie (dubel z klasą Konto?)
19. *string send\_message(void);* - pozwala użytkownikowi wysłac wiadomość do pracownika
20. *void setting\_treshold(void);* - interfejs zmiany progu wraz z instrukcjami postepowania
21. *void modifying\_account(void);* - interfejs zmian cech rozpoznawalnych konta wraz   
    ze stosownymi instrukcjami postępowania
22. *Firma* – klasa pochodna klasy *Konto* – klasa kont firmowych, wyróżnia się możliwością posiadanie wielu współ-posiadaczy (wspólników), którzy mają równe prawa I równe głosy w zarządzaniu nim.
23. *string get\_nazwa(void) const*; - zwraca nazwę firmy
24. *long double get\_saldo(void);* - zwraca saldo firmy
25. *long double get\_prog\_admin(void);* - zwraca próg operacji nieautoryzowanych przez pracowników banku
26. *long double get\_prog\_wspolnik(void);* - zwraca aktualny próg operacji nieautoryzowanych przez pozostałych wspólników

Słowo wyjaśnienia: założyliśmy, że wspólnicy sobie nie ufają całkowicie, więc dajemy im możliwość kontroli współdostępowiczów poprzez ustawienie limitu, od którego wszyscy muszą zatwierdzić przelew środków;

Różnice vs prog\_admin: mogą chciec sami sie kontrolowac, ale nie chciec interwencji pracowników banku, dlatego są to dwa osobne progi

1. *map<string, string> get\_passwords(void);* - zwraca hasła dostępu wspólników
2. *map <string, string> get\_imie(void);* - zwraca imiona wspólników
3. *map <string, bool> get\_zgody(void);* - zwraca, czy odbywa sie głosowanie, a jeśli tak, to kto I jakie oddał głosy
4. *long double get\_propozycja(void);* - zwraca jaka nowa kwota jest przegłosowywana
5. *int get\_opcja(void);* - zwraca jakie głosowanie aktualnie sie odbywa
6. *void set\_imie(map<string, string> &imie);* - daje możliwość zmian imion wspólników
7. *void set\_passwords(map<string, string> &passwords);* - daje możliwość zmiany haseł wspólników
8. *void set\_nazwa(string &nazwa);* - daje możliwość zmiany nazwy
9. *void set\_saldo(long double &saldo);* - ustawia nową wartość konta
10. *void set\_prog\_admin(long double &prog\_admin);* - ustawia przegłosowany nowy próg
11. *void set\_prog\_wspolnik(long double &prog\_wspolnik);* - ustawia przegłosowany nowy próg
12. *void set\_zgody(map<string,bool> &zgody);* - pozwala na wprowadzenie zmian do mapy zgód wyrażonych przez wspólników;
13. *void set\_propozycja(long double &propozycja);* - ustawia wartość kwoty zaproponowanej przez tworzącego głosowanie
14. *void set\_opcja(int& opcja);* - ustawia flagę, która rozpoznaje, o zmianę czego toczy się głosowanie
15. *string send\_message(void);* - wysyła wiadomość do pracownika banku
16. *void ustawianie\_progu(Firma &obiekt, string imie, string ksywa);* - pozwala użytkownikowi (o loginie ksywa I imieniu imie) rozpocząć głosowanie nad zmianą jednego z progów dla firmy obiekt
17. *bool logowanie(void);* - metoda do logowania
18. *Fundacja* – klasa pochodna klasy *Konto*, jest niemal identycznie skonstruowana, jak klasa Firma; różnica polega na przejrzystości finansów i jej ogólnodostępności – nie trzeba być pracownikiem Fundacji aby móc śledzić jej stan finansowy
19. *string get\_nazwa(void) const;* - zwraca nazwę firmy
20. *long double get\_saldo(void);* - zwraca saldo firmy
21. *long double get\_prog\_admin(void);* - zwraca próg operacji nieautoryzowanych przez pracowników banku
22. *long double get\_prog\_wspolnik(void);* - zwraca aktualny próg operacji nieautoryzowanych przez pozostałych wspólników
23. *map<string, string> get\_passwords(void);* - zwraca hasła dostępu wspólników
24. *map <string, string> get\_imie(void);* - zwraca imiona wspólników
25. *map <string, bool> get\_zgody(void);* - zwraca, czy odbywa sie głosowanie, a jeśli tak, to kto I jakie oddał głosy
26. *long double get\_propozycja(void);* - zwraca jaka nowa kwota jest przegłosowywana
27. *int get\_opcja(void);* - zwraca jakie głosowanie aktualnie sie odbywa
28. *void set\_imie(map<string, string> &imie);* - daje możliwość zmian imion wspólników
29. *void set\_passwords(map<string, string> &passwords);* - daje możliwość zmiany haseł wspólników
30. *void set\_nazwa(string &nazwa);* - daje możliwość zmiany nazwy
31. *void set\_saldo(long double &saldo);* - ustawia nową wartość konta
32. *void set\_prog\_admin(long double &prog\_admin);* - ustawia przegłosowany nowy próg
33. *void set\_prog\_wspolnik(long double &prog\_wspolnik);* - ustawia przegłosowany nowy próg
34. *void set\_zgody(map<string,bool> &zgody);* - pozwala na wprowadzenie zmian do mapy zgód wyrażonych przez wspólników;
35. *void set\_propozycja(long double &propozycja);* - ustawia wartość kwoty zaproponowanej przez tworzącego głosowanie
36. *void set\_opcja(int& opcja);* - ustawia flagę, która rozpoznaje, o zmianę czego toczy się głosowanie
37. *string send\_message(void);* - wysyła wiadomość do pracownika banku
38. *void ustawianie\_progu(Fundacja &obiekt, string imie, string ksywa);* - pozwala użytkownikowi (o loginie ksywa I imieniu imie) rozpocząć głosowanie nad zmianą jednego z progów dla fundacji obiekt
39. *bool logowanie(Fundacja &obiekt);* - metoda do logowania; tutaj wyjątkowo argumentem funkcji jest obiekt klasy Fundacja ze względu na opisaną powyżej wyjątkowość tej klasy, tj przejrzystość I dostęp informacji dla postronnych. Po wpisaniu ID fundacji bez hasła użytkownik otrzymuje informacje o stanie finansowym Fundacji bez wchodzenia do menu zarządzania.
40. Struktura *powiadomienia*: zawiera dwa pola – level I tresc, przy jej użyciu przypisywane są zadania pracownikom oraz administratorom. Pola oznaczają kolejno priorytet I opis zadania.
41. *Pracownik* – klasa pochodna klasy *Konto*, klasa przeznaczona dla szeregowego pracownika banku, który nie ma możliwości zarządzania systemem z punktu widzenia programowgo.
42. *string get\_nazwisko(void) const;* - zwraca nazwisko
43. *vector <powiadomienia> get\_notifications(void);* - zwraca wektor zadań dla pracowników
44. *void set\_nazwisko(const std::string& nazwisko);* - metoda odpowiedzialna za zmiane nazwiska właściciela konta
45. *void set\_notifications(powiadomienia notification);* - metoda odpowiedzialna za tworzenie wektora z zadaniami
46. *void send\_message(void);* - wysyła wiadomość
47. *string show\_notifications\_prac(void);* - pokazuje przypisane konkretnemu obiektowi zadanie
48. *Admin* – klasa pochodna *klasy Pracownik* - obdarzona największymi możliwościami. Administrator zarządza całym systemem. W jego kompetencjach leży potwierdzanie tworzenia, odblokowywania i usuwania kont czy zatwierdzanie przelewów.
49. *bool transfer\_approval(powiadomienia notification);* - panel potwierdzania przelewów
50. *bool give\_back\_control(powiadomienia notification);* - panel zdejmowania blokady z konta
51. *bool creating\_account\_approval(powiadomienia notification);* - panel zatwierdzania tworzenia konta
52. *bool deleting\_account\_approval(powiadomienia notification);* - panel zatwierdzania usuwania konta
53. *bool modifying\_account\_approval(powiadomienia notification);* - panel zatwierdzania modyfikowania konta
54. *string show\_notifications\_adm(void);* - metoda odpowiedzialna za pokazanie przypisanego zadania dla obiektu

# Zasoby

W projekcie wykorzystywane są następujące pliki zasobów:

*Obiekty\_klienci.txt*

*Obiekty\_firmy.txt*

*Obiekty\_fundacje.txt*

*Obiekty\_pracownicy.txt*

*Obiekty\_admini.txt*

Powyższe pliki tekstowe zawierają wszystkie informacje o dotychczas stworzonych (nieusuniętych) obiektach z klas Klient, Firma, Fundacja. Struktura tych plików jest taka sama: na każdą linię przypadają informacje o dokładnie jednym obiekcie. Każda informacja jest rozdzielona separatorem ‘|’, a zakończenie linii sygnalizowane jest kolejką znaków ‘ |end| ‘. Informacje są przechowywane w kolejności zdefiniowanej przez konstruktor każdej klasy. Dodatkowo klasy o składowych z biblioteki map są rozdzielone między sobą separatorami ‘%’, ‘#’, ‘$’. Przykład klienta zapisanego w ww sposób:

111|pass|michal|szczepaniak|0|1000|kdsjfbdksgbkjb|end|

gdzie kolejne informacje to: ID, hasło, imię, nazwisko, saldo, próg transakcji nieautoryzowanych, wiadomość.

W programie zostały stworzone algorytmy „szyfrujące” i „deszyfrujące” strukturę zapisu w ramach funkcji pobierających oraz wpisujących dane z/do pliku.

# Dalszy rozwój i ulepszenia

1. W klasach Firma i Fundacja zaimplementować różną wagę głosów poszczególnych wspólników umożliwiając szefostwu dawanie dostępu do konta pracownikom o ograniczonych możliwościach.
2. Rejestr dokonanych transakcji (najlepiej w bazie danych)
3. Oprogramowanie dla bankomatu zsynchronizowane z oprogramowaniem bankowym

# Inne

W programie nie zostało zaimplementowane wyświetlanie polskich znaków, wszystkie instrukcje pojawiające się w konsoli są zapisane z użyciem alfabetu nie zawierającego polskich znaków specjalnych.