Projekt zaliczeniowy

Programowanie obiektowe
Rok akademicki 2024/2025

Autorzy:
Kacper Łapot
Aleksander Jasiński
Jakub Kaźmierczyk
Rafał Łubkowski

Aplikacja “myBank”

Nasz Projekt to kompleksowa symulacja aplikacji bankowej, w której (jako użytkownik) możemy dokonywać wpłat oraz wypłat jak i robić przelewy. Dla administratorów przewidziano dodatkowe funkcjonalności, takie jak tworzenie i usuwanie kont użytkowników oraz zarządzanie pracownikami (dodawanie i usuwanie). Aplikacja jest w pełni funkcjonalna: posiada obsługę wyjątków, ma własny stworzony przez nas interfejs użytkownika oraz wszystkie zmiany są zapisywane do pliku XML.

Podział ról

1. **Kacper Łapot**:
    - podział ról
    - koordynowanie działań całego projektu
    - kod funkcjonalności GUI

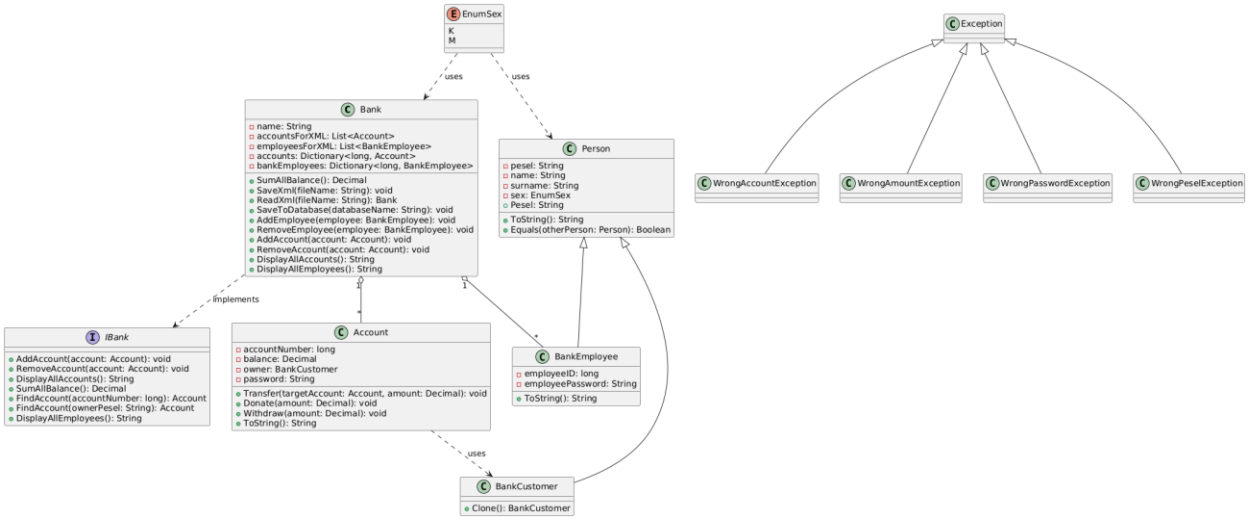
2. **Aleksander Jasiński**:
    - zapis do bazy danych
    - testy jednostkowe
    - szata graficzna interfejsu użytkownika

3. **Jakub Kaźmierczyk**:
    - klasa i interfejs Banku
    - obsługa wyjątków

4. **Rafał Łubkowski**:
    - klasy Person, BankCustomer, BankEmployee oraz Account
    - schemat UML

Dla każdej klasy należy dodać jej opis, podać jej rolę w projekcie (uzasadnić potrzebę jej stworzenia), uzasadnić modyfikatory dostępu do pól, itp.
Ważne! Hasło przy tworzeniu konta musi mieć conajmniej 5 znaków – jedną dużą literę, jedną małą i jedną cyfrę

Diagram klas



Klasy:

1. **Account.cs**
    - **Przeznaczenie**: Klasa reprezentuje konto bankowe. Może zawierać informacje o numerze konta, saldzie, transakcjach oraz metody związane z operacjami na koncie, np. wpłaty, wypłaty czy przelewy.
2. **Bank.cs**
    - **Przeznaczenie**: Klasa główna reprezentująca bank. Zapewne zarządza kolekcją klientów, pracowników i kont bankowych. Może oferować metody do logowania użytkowników, zakładania kont i zarządzania całym systemem.
3. **BankCustomer.cs**
    - **Przeznaczenie**: Klasa reprezentująca klienta banku. Dziedziczy po klasie abstrakcyjnej `Person`. Może zawierać dodatkowe informacje specyficzne dla klientów (np. listę kont bankowych przypisanych do klienta) oraz metody do interakcji z kontem.
4. **BankEmployee.cs**

- **Przeznaczenie**: Klasa reprezentująca pracownika banku. Dziedziczy po klasie abstrakcyjnej `Person`. Może zawierać funkcjonalności specyficzne dla pracowników, np. obsługę klientów, zarządzanie kontami czy rozwiązywanie problemów technicznych.

5. **IBank.cs**
   - **Przeznaczenie**: Interfejs definiujący kontrakt dla klasy `Bank`. Może zawierać deklaracje metod takich jak rejestracja użytkowników, obsługa kont czy raportowanie, zapewniając spójność w implementacji.
6. **Person.cs**
   - **Przeznaczenie**: Abstrakcyjna klasa bazowa dla `BankCustomer` i `BankEmployee`. Może przechowywać wspólne informacje o osobach, takie jak imię, nazwisko, PESEL oraz metody wspólne dla klientów i pracowników (np. autoryzacja).
7. **Program.cs**
   - **Przeznaczenie**: Główna klasa programu, zawierająca metodę `Main`. Jest punktem wejściowym aplikacji, odpowiedzialnym za uruchomienie programu i inicjalizację elementów systemu bankowego.
8. **WrongAccountException.cs**
   - **Przeznaczenie**: Klasa reprezentująca wyjątek rzucany w sytuacji, gdy wystąpi problem z kontem bankowym (np. nieprawidłowy numer konta lub brak dostępu do niego).
9. **WrongAmountException.cs**
   - **Przeznaczenie**: Klasa wyjątku używana w przypadku nieprawidłowej kwoty podczas operacji bankowych, np. przy próbie wpłaty ujemnej kwoty lub wypłaty przekraczającej saldo.
10. **WrongPasswordException.cs**
    - **Przeznaczenie**: Klasa reprezentująca wyjątek rzucany, gdy użytkownik poda błędne hasło podczas logowania lub autoryzacji.
11. **WrongPeselException.cs**
    - **Przeznaczenie**: Klasa wyjątków związana z błędami w numerze PESEL (np. nieprawidłowy format, długość lub brak zgodności z danymi użytkownika).

Wykonaliśmy również szereg testów jednostkowych, aby mieć pewność, że wszystkie funkcjonalności działają poprawnie. Dzięki temu mogliśmy automatycznie zweryfikować poprawność działania kluczowych elementów aplikacji, takich jak operacje na kontach bankowych, logowanie użytkowników oraz obsługa wyjątków.

Testy jednostkowe pozwoliły na sprawdzenie, czy metody odpowiadające za wpłaty, wypłaty i przelewy działają zgodnie z założeniami, a saldo konta jest aktualizowane prawidłowo. Dodatkowo zweryfikowaliśmy obsługę wyjątków, takich jak `WrongAmountException` czy

`WrongPasswordException`, co pozwoliło upewnić się, że system reaguje właściwie na nieprawidłowe dane wejściowe.

Opis funkcjonalności

Kod składa się z kilku klas (w tym klas wyjątków), cały schemat klas jest stworzony w pliku UML.

Opis poszczególnych funkcji (i metod) w różnych klasach:

Klasa Account – klasa należącą do użytkownika (BankCustomer):

public void Transfer(Account a1, decimal amount) – funkcja wykonywująca przelew między kontami

public void Donate(decimal amount) – dokonywanie wpłat na konto

public void Withdraw(decimal amount) – wykonywanie wypłat z konta

Klasa Bank – klasa należącą do administratora (BankEmployee):

public void AddEmployee(BankEmployee employee) – dodawanie pracownika z banku

public void RemoveEmployee(BankEmployee employee) – usuwanie pracownika z banku

public BankEmployee? FindEmployee(string password) – wyszukiwanie pracownika z banku

public string DisplayAllEmployess() – wypisywanie wszystkich pracowników z banku

public void AddAccount(Account account) – dodawanie konta do banku

public void RemoveAccount(Account account) – usuwanie konta z banku

public void RemoveAccountByPeselAndPassword(string pesel, string password) – usuwanie konta używając peselu i hasła

public string DisplayAllAccounts() – wyświetlenie wszystkich kont

public Account FindAccount() – szukanie konta po numerze bankowym lub haśle

public void SaveXml(string fileName) – zapis do pliku XML

public Bank ReadXml(string fileName) – odczyt z pliku XML

public void saveToDatabase(string connectionString) – zapis do bazy danych

public void Sort() – sortowanie listy kont po balansie

public static void CreateBank() – inicjalizowanie instancji banku

IBank – interfejs przechowujący prototypy funkcji klasy Bank

Person – klasa, po której dziedziczą BankCustomer oraz BankEmployee

Instrukcja Obsługi:

1. Po uruchomieniu projektu (plik z GUI WPF) widoczne jest pare guzików:
    - ***Login*** – guzik przekierowywujący nas do strony z logowaniem
    - ***About us*** – wyświetla sprawozdanie
    - ***Exit*** – kończy działanie programu
2. Po kliknięciu ***Login*** wyświetla nas się nowa strona z logowaniem, do którego musimy podać hasło – zalogować możemy się zarówno jako pracownik jaki i użytkownik:
    - **Hasła użytkownika:** Barcelona11, Krajewska99, BorkowskaL329, MiLas84, SIkoraaa32, KacSzym203, kNowak1
    - **Hasła administratora:** HasloPracownika1, HasloPracownika2
3. Po zalogowaniu się pojawia się jedno z dwóch okien (w zależności od wybranego hasła):
    - W momencie zalogowania się jako użytkownik wyświetlają się nasze dane – imię, nazwisko, balans. Możemy wykonać kilka operacji używając guzików – ***Deposit, WIthdraw, Transfer*** (dla ułatwienia jako numer konta podajemy numer bez zer wiodących)
    - Jeśli zalogujemy się jako administrator wyświetlają się baza wszystkich użytkowników należących do banku oraz ich dane – imię i nazwisko, numer konta, balans. Również w tym przypadku możemy wykonać kilka operacji: ***Add Account, Remove Account, Add Employee, Remove Employee***, a także ***Sort*** - funkcję, która umożliwia sortowanie kont według balansu.
4. Na obu panelach w prawym dolnym rogu znajduje się przycisk: ***Log out and save***, który powoduje powrót do strony logowania oraz zapisanie zmian w pliku XML.