

Projekt zespołowy – aplikacja bankomatu

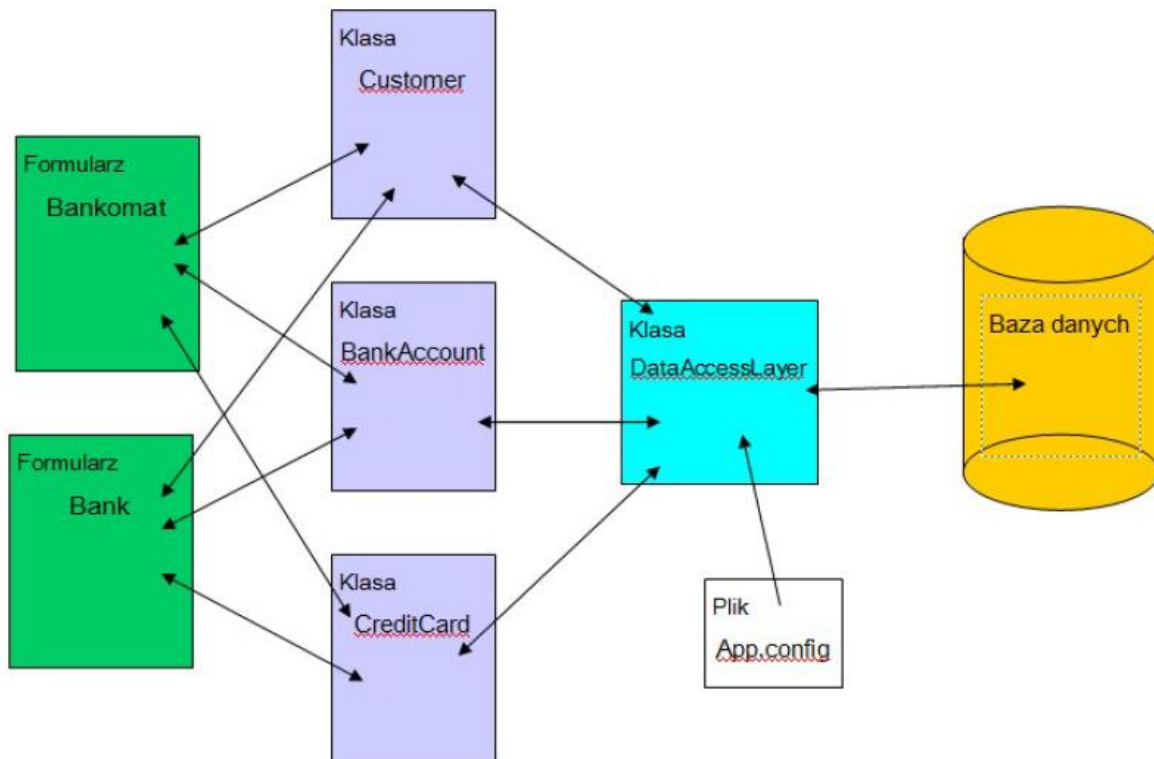
Spis treści

Projekt zespołowy – aplikacja bankomatu.....	1
Wstęp	2
Schemat aplikacji	2
Warstwa Bazy Danych.....	2
Warstwa DataAccessLayer	3
Warstwa Aplikacji	5
Warstwa Formularza	8
Modele wytwarzania oprogramowania.....	13
Osoby i role w projekcie	14
Diagramy działania aplikacji	15
Use Case Diagram.....	15
Activity Diagram	16
Sequence Diagram	17
Class Diagram	18
Podsumowanie	18

Wstęp

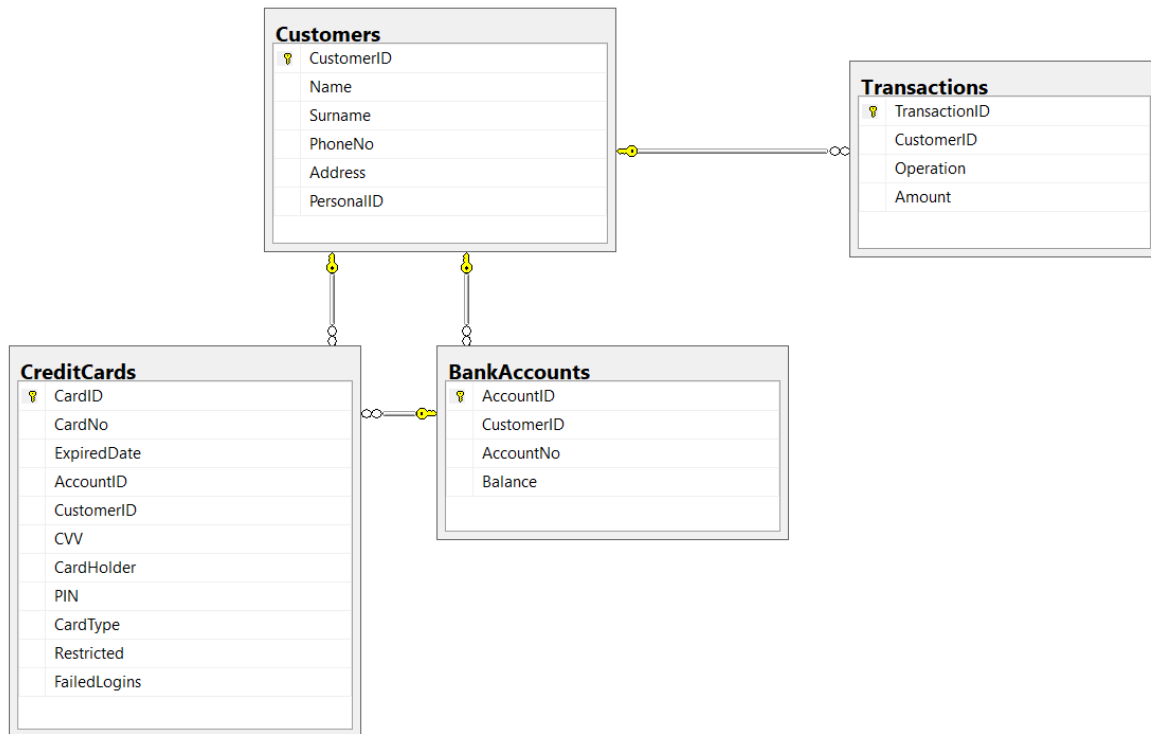
Projekt ma za zadanie stworzenie aplikacji symulującej działanie bankomatu. Aplikacja składa się z 4 warstw: warstwy bazy danych, warstwy dostępu do danych (DataAccessLayer), warstwy aplikacji i warstwy formularza.

Schemat aplikacji



Warstwa Bazy Danych

Warstwa bazy danych zawiera jedną bazę składającą się z 4 tabel: **Customer**, **BankAccounts**, **CreditCards** oraz **Transactions**. Poniżej widoczny jest schemat bazy danych wraz z powiązaniem między tabelami.



Warstwa DataAccessLayer

Warstwa dostępu do danych - **DataAccessLayer** - pozwala na dostęp aplikacji do bazy danych.

Umożliwia to globalny obiekt `sqlConnection`. Łańcuch połączenia z bazą danych (`connectionString`) znajduje się pliku konfiguracyjnym `App.config`. Odczytanie łańcucha połączenia z pliku konfiguracyjnego aplikacji umożliwia klasa `ConfigurationManager` znajdująca się w przestrzeni nazw `System.Configuration`.

Klasa `DataAccessLayer` zawiera ponadto 5 publicznych metod, które wystarczają do obsługi bazy danych.

Metoda `selectData(SqlCommand sqlCommand)` obsługuje zapytania typu *select* i zwraca wynik tego zapytania w tabeli danych (`DataTable`).

Metoda `selectReader(sqlCommand sqlCommand)` również obsługuje zapytania typu *select*, ale zwraca wynik w postaci strumienia wierszy (`SqlDataReader`). Metodę tę można wykorzystać do znalezienia pojedynczej wartości w bazie danych.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.SqlClient;
using System.Data;
using System.Configuration;

namespace WinFormBankomat_N_19
{
    36 references
    class DataAccessLayer
    {
        SqlConnection sqlConnection =
            new SqlConnection(ConfigurationManager.ConnectionStrings["abcd"].ConnectionString);

        3 references
        public DataTable selectData(SqlCommand sqlCommand)
        {
            //obsługa zapytań typu select
            // otwarcie i zamykanie połączenia z bazą danych wykonuje obiekt sda typu SqlDataAdapter
            sqlCommand.Connection = sqlConnection;
            SqlDataAdapter sda = new SqlDataAdapter(sqlCommand);
            DataTable dt = new DataTable();
            sda.Fill(dt);
            return dt;
        }

        11 references
        public SqlDataReader returnReader(SqlCommand sqlCommand)
        {
            //obsługa zapytań select, zwraca dane w postaci strumienia wierszy
            // wymagane jest otwarte połączenie z bazą danych
            sqlCommand.Connection = sqlConnection;
            SqlDataReader reader = sqlCommand.ExecuteReader();
            return reader;
        }

        3 references
        public SqlConnection getConn()
        {
            return sqlConnection;
        }
    }
}

```

Metody `connectionOpen()` oraz `connectionClose()` pozwalają na otwieranie i zamykanie połączenia z bazą danych.

Metoda `queryExecution(sqlCommand sqlCommand)` służy do wykonywania zapytań typu *insert*, *update* i *delete*. Metodę tę wykorzystuje się do aktualizacji salda na rachunku czy wprowadzania danych nowego klienta, rachunku lub karty.

```

public SqlConnection getConn()
{
    return sqlConnection;
}

8 references
public void queryExecution(SqlCommand sqlCommand)
{
    // wymagane jest otwarte połączenie z bazą danych
    // obsługa zapytań typu insert, update i delete
    sqlCommand.Connection = sqlConnection;
    sqlCommand.ExecuteNonQuery();
}

20 references
public bool connectionOpen()
{
    try
    {
        sqlConnection.Open();
        return true;
    }
    catch
    {
        return false;
    }
}

26 references
public bool connectionClose()
{
    try
    {
        sqlConnection.Close();
        return true;
    }
    catch
    {
        return false;
    }
}
}

```

Warstwa Aplikacji

Warstwa aplikacji składa się z 4 klas: **Customer**, **CreditCard**, **BankAccount** i **Transaction**.

Metody tych klas związane są z odpowiadającymi im tabelami w bazie danych. Wszystkie kolumny tabel odpowiadają właściwościom ich klas. Ponadto zawierają one szereg metod umożliwiających odczyt i zapis danych do bazy.

Właściwości klasy **Customer** odpowiadają kolumnom w tabeli **Customers**.

```
{
    12 references
    class Customer
    {
        2 references
        public int CustomerID { get; private set; }
        8 references
        public string Name { get; set; }
        8 references
        public string Surname { get; set; }
        4 references
        public long PhoneNo { get; private set; }
        4 references
        public string Address { get; private set; }
        8 references
        public long PersonalID { get; set; }
    }
}
```

Update

Script File:

dbo.Customers.sql

	Name	Data Type	Allow Nulls	Default	
no	CustomerID	int	<input type="checkbox"/>		
	Name	varchar(20)	<input type="checkbox"/>		
	Surname	varchar(30)	<input type="checkbox"/>		
	PhoneNo	bigint	<input type="checkbox"/>		
	Address	varchar(50)	<input type="checkbox"/>		
	PersonalID	bigint	<input type="checkbox"/>		
			<input type="checkbox"/>		

Keys (1)

<unnamed> (Primary Key, Clustered: CustomerID)

Check Constraints (0)

Indexes (0)

Foreign Keys (0)

Triggers (0)

Design

T-SQL

```
CREATE TABLE [dbo].[Customers] (  
[CustomerID] INT IDENTITY (1, 1) NOT NULL,  
[Name] VARCHAR (20) NOT NULL,  
[Surname] VARCHAR (30) NOT NULL,  
[PhoneNo] BIGINT NOT NULL,  
[Address] VARCHAR (50) NOT NULL,  
[PersonalID] BIGINT NOT NULL,  
PRIMARY KEY CLUSTERED ([CustomerID] ASC)  
);
```

Oprócz właściwości, w klasie **Customer** znajdują się 2 publiczne metody **getCustomerID(long pesel)** oraz **getCustomerInfo(long pesel)**, które jako argument przyjmują nr PESEL. Pierwsza metoda zwraca wartość klucza głównego z tabeli **Customers** szukanego klienta, druga ustawia właściwości obiektu typu **Customer**. Obie metody korzystają z obiektu **sqlCommand** przypisując jego właściwości **sqlCommand.CommandText** odpowiednie zapytanie typu *select*, a następnie wywołują metodę **dal.returnReader(sqlCommand)** na obiekcie **dal** typu **DataAccessLayer**. Metoda

returnReader(sqlCommand) zwraca obiekt typu SqlDataReader (reader), gdzie dostęp do poszczególnych wierszy uzyskuje się poprzez wywołanie metody read().

```
public int getCustomerID(long pesel)
{
    string query = "select CustomerID from Customers where PersonalID = @pesel";
    SqlCommand sqlCommand = new SqlCommand();
    sqlCommand.CommandText = query;
    sqlCommand.Parameters.AddWithValue("@pesel", pesel);
    DataAccessLayer dal = new DataAccessLayer();

    if (dal.connectionOpen())
    {
        SqlDataReader reader = dal.returnReader(sqlCommand);
        if (reader.HasRows)
        {
            reader.Read(); // czyta 1 wiersz
            this.CustomerID = Convert.ToInt32(reader[0].ToString());
            reader.Close();
            dal.connectionClose();
            return this.CustomerID;
        }
        else
        {
            dal.connectionClose();
            return -1; // brak klienta w bazie banku
        }
    }
    else return -2; // brak połączenia z bazą danych
}
```

```
public int getCustomerInfo(long pesel)
{
    string query = "select Name, Surname, PhoneNo, Address, PersonalID from Customers where PersonalID = @pesel";
    SqlCommand sqlCommand = new SqlCommand();
    sqlCommand.CommandText = query;
    sqlCommand.Parameters.AddWithValue("@pesel", pesel);
    DataAccessLayer dal = new DataAccessLayer();

    if(dal.connectionOpen())
    {
        SqlDataReader reader = dal.returnReader(sqlCommand);
        if(reader.HasRows)
        {
            reader.Read();
            this.Name = reader[0].ToString();
            this.Surname = reader[1].ToString();
            this.PhoneNo = Convert.ToInt64(reader[2].ToString());
            this.Address = reader[3].ToString();
            this.PersonalID= Convert.ToInt64(reader[4].ToString());
            reader.Close();
            dal.connectionClose();
            return 1; // klient został znaleziony
        }
        else
        {
            dal.connectionClose();
            return -1; // klient nie został znaleziony
        }
    }
    else
    {
        return -2; // brak połączenia z bazą danych
    }
}
```

Ponadto, w klasie `Customer` znajduje się publiczna metoda `addCustomer()`, która pobiera dane z formularza. Jak wcześniej, metoda korzysta z obiektu `SqlCommand` przypisując jego właściwości `SqlCommand.CommandText` zapytanie typu *insert*, a następnie wywołuje metodę `dal.queryExecution(sqlCommand)`.

```
public string addCustomer()
{
    string query = "insert into Customers values(@name,@surname,@phone,@address,@pesel)";
    SqlCommand sqlCommand = new SqlCommand();
    sqlCommand.CommandText = query;
    sqlCommand.Parameters.AddWithValue("@name", this.Name);
    sqlCommand.Parameters.AddWithValue("@surname", this.Surname);
    sqlCommand.Parameters.AddWithValue("@phone", this.PhoneNo);
    sqlCommand.Parameters.AddWithValue("@address", this.Address);
    sqlCommand.Parameters.AddWithValue("@pesel", this.PersonalID);
    DataAccessLayer dal = new DataAccessLayer();

    try
    {
        dal.connectionOpen();
        dal.queryExecution(sqlCommand);
        dal.connectionClose();

        return "new customer inserted";
    }
    catch(Exception ex)
    {
        return ex.ToString();
    }
}
```

Form1

	CustomerID	Name	Surname	PhoneNo	Address	PersonalID
	1	Jan	Kowalski	224568596	00-950 Warszaw...	75112344444
	2	Adam	Nowak	600231456	05-500 Piaseczn...	84062811111
▶	3	Name	Surname	123456789	adress	12345
*						

customer info new customer

Name

Surname

Phone

Address

Pesel

add customer

new customer inserted

Warstwa Formularza

Warstwa Formularza składa się z formularza Bankomat, który ma za zadanie symulować działanie bankomatu, oraz formularza Bank, w pewnym sensie symulujący działanie okienka kasowego w banku.



Użytkownik ma możliwość wyboru, czy chce zalogować się do aplikacji jako Klient czy jako Pracownik. W przypadku wyboru logowania jako Klient, aplikacja umożliwi wpisanie nr karty i kodu PIN w odpowiednie pola edycyjne oraz zatwierdzenie operacji przyciskiem.

Form2

Logowanie

Numer karty

Kod PIN

Zaloguj Powrót

Aplikacja umożliwia zasymulowanie sytuacji, w której karta jest przeterminowana, zablokowana lub skradziona, co oznajmi odpowiednim komunikatem. Trzykrotne wpisanie błędnego PINu również skutkuje zablokowaniem karty i wyświetleniem odpowiedniego komunikatu.

Form2

Logowanie

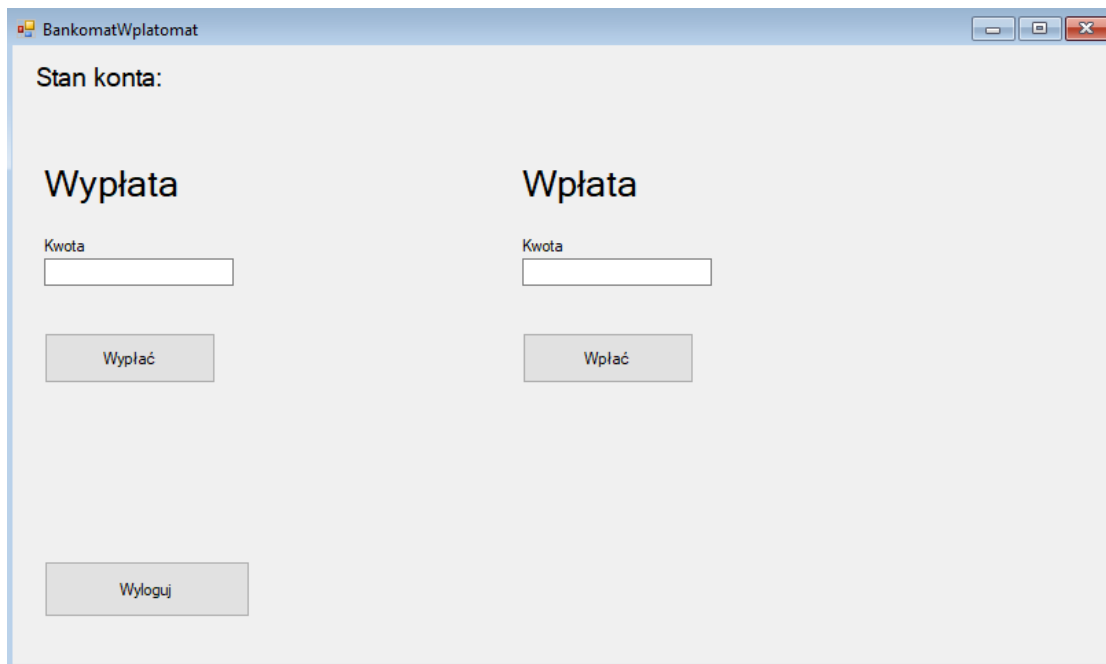
Numer karty

Kod PIN

Zbyt wiele błędnych logowań, karta zablokowana

Zaloguj Powrót

Po zalogowaniu jako Klient, użytkownik ma dostęp do informacji o stanie środków na koncie oraz ma możliwość wypłaty i wpłaty gotówki.



BankomatWpłatomat

Stan konta:

Wyplata

Kwota

Wyplac

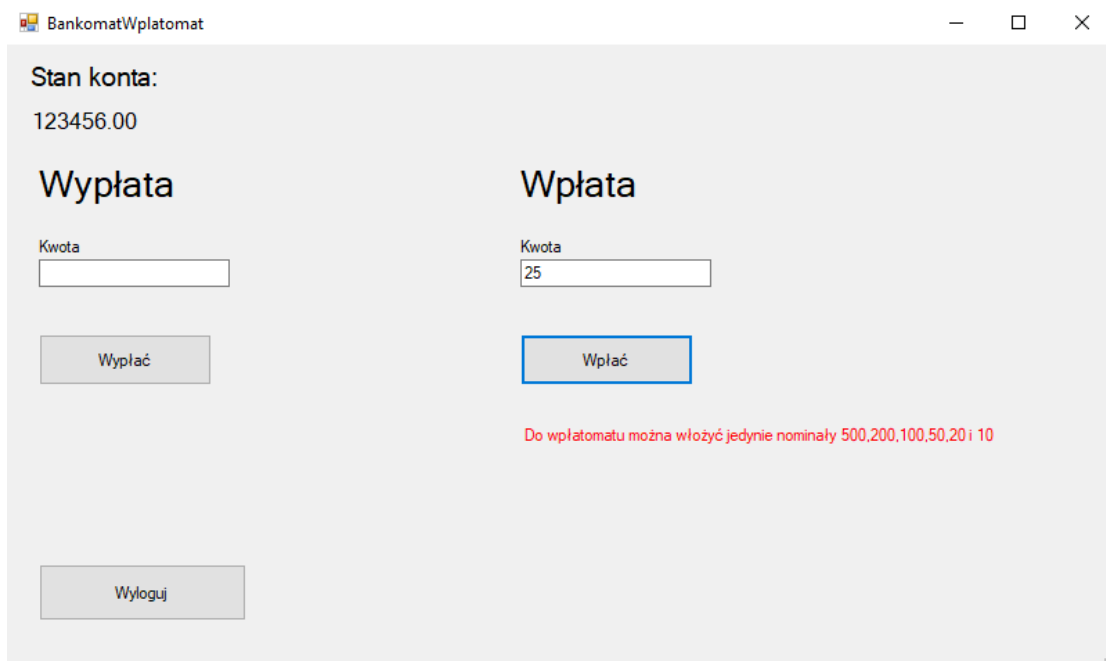
Wplata

Kwota

Wplac

Wyloguj

System pozwala na wypłaty i wpłaty wyłącznie banknotów, a więc kwota, jaką chcemy wpłacić lub wypłacić musi być możliwa do zaprezentowania jako suma nominałów. Nie spełnienie tego warunku skutkuje wyświetleniem odpowiedniego komunikatu. Również próba wypłacenia większej ilości pieniędzy, niż jest na koncie, jest niemożliwa i skutkuje wyświetleniem informacji z błędem.



BankomatWpłatomat

Stan konta:
123456.00

Wyplata

Kwota

Wyplac

Wplata

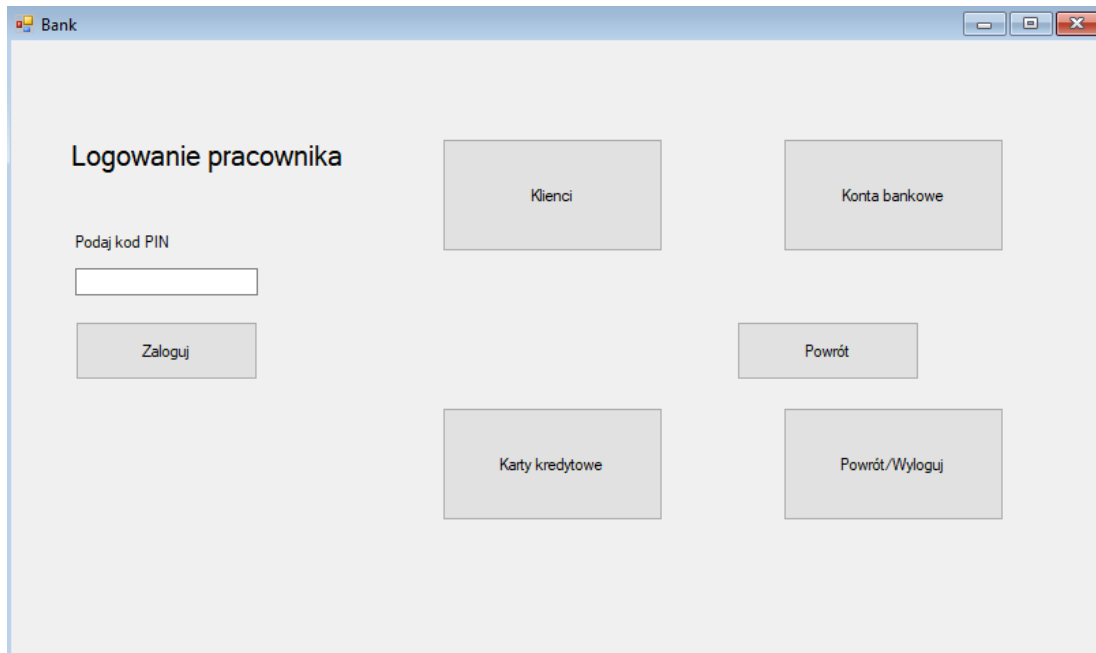
Kwota

Wplac

Do wpłatomatu można włożyć jedynie nominały 500,200,100,50,20 i 10

Wyloguj

Zalogowanie jako Pracownik pozwala na uzyskanie szeregu informacji o Klientach, kontach bankowych i kartach kredytowych.



Opcja „Klienci” pozwala na szukanie Klientów według różnych kryteriów oraz na dodawanie nowych Klientów.

Form1

	CustomerID	Name	Surname	PhoneNo	Address	PersonalID
▶	1	Jan	Kowalski	224568596	00-950 Warszaw...	75112344444
	2	Adam	Nowak	600231456	05-500 Piaseczn...	84062811111
*						

customer info new customer

PESEL 75112344444 1

Search Customer

Search Customer

Powrót

Customer
Name = Jan
Surname = Kowalski
Phone = 224568596
Address = 00-950 Warszawa, Woronicza 15 m 23
PersonalID = 75112344444

Opcja „Konta bankowe” umożliwia wyszukiwanie informacji o istniejących kontach, dodawanie nowych kont oraz dokonywanie wypłat i wpłat u bankiera.

FormAccounts

	AccountNo	Balance	PersonalID
▶	53163171234	362734.00	75112344444
	64712478256	123456.00	84062811111
*			

Informacje o koncie Dodaj nowe konto Wpłata/Wypłata u bankiera

PESEL 75112344444 1

Znajdź Konto

Znajdź Konto

Powrót

Customer
Name = Jan
Surname = Kowalski
PersonalID = 75112344444
AccountNo = 53163171234
Balance = 362734

Opcja „Karty kredytowe” wyświetla informacje o kartach kredytowych, umożliwia szukanie konkretnych kart według kryteriów wyszukiwania oraz na dodawanie nowych kart.

FormCreditCards

	CardID	CardNo	ExpiredDate	AccountID	CustomerID	CVV	CardH
▶	1	1234	6/15/2020	1	1	123	123
	3	4321	6/16/2022	2	2	123	123
*							

Powrót

Informacje o karcie Dodaj kartę

CardNo

Customer -- Wybierz Klienta-- ▾

Account

CVV

CardHolder

CardType

Dodaj kartę

Modele wytwarzania oprogramowania

W procesie tworzenia oprogramowania możemy wyróżnić dwa podstawowe modele: model kaskadowy oraz model cykliczny. W obu modelach możemy wyróżnić następujące kroki: planowanie, analiza, projektowanie, implementacja (development), testowanie oraz utrzymywanie. Model kaskadowy polega na wykonywaniu podstawowych czynności jako odrębnych faz projektowych, kolejno po sobie. Jeśli któraś z faz zwróci niesatysfakcjonujący produkt cofamy się wykonując kolejne iteracje aż do momentu kiedy otrzymamy satysfakcjonujący produkt. W modelu cyklicznym wykonujemy następujące kroki w cyklach, aż do momentu otrzymania zadowalającego produktu. Na każdym etapie możliwa jest rewizja procesu i dostosowanie produktu do ciągle zmieniających się potrzeb i uwag Klienta.

Osoby i role w projekcie

Zadania do wykonania w projekcie zostały przydzielone do konkretnych osób ze względu na ich preferencje oraz na posiadane umiejętności. Każda osoba z grupy pełni określoną rolę w projekcie:

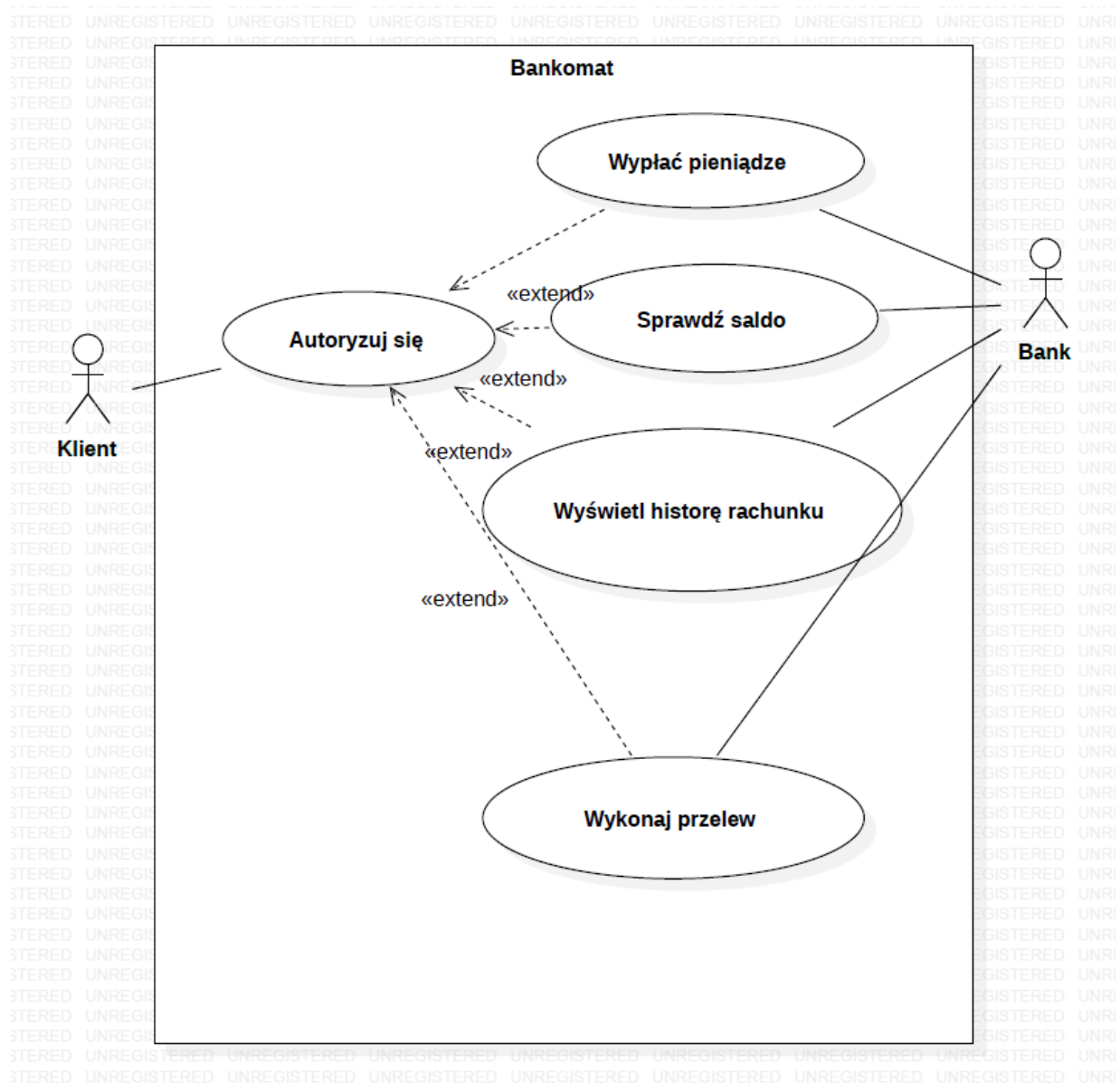
1. Kamil Prokopiuk – Project Manager – zarządzanie projektem, przydzielanie zadań, przegląd i kontrola zmian w kodzie, sporządzanie dokumentacji.
2. Adam Szymański – Analityk – dba o to aby projekt spełniał wymagania biznesowe i osiągał zamierzone cele, istotne wsparcie w trakcie wybierania rozwiązań, określania celów i nadania projektowi odpowiedniego kierunku, również sporządzanie dokumentacji w postaci diagramów.
3. Hanna Kasprzak – Architekt baz danych – projekt, implementacja i utrzymanie bazy danych spełniającej wymagania aplikacji, stworzenie schematu bazy danych.
4. Artur Rosiński – UI/UX Designer – osoba odpowiedzialna na stworzenie nowoczesnego i intuicyjnego GUI aplikacji.
5. Sylwester Kwiatkowski – Programista i Architekt Systemu – jedna z najważniejszych osób w zespole, najbardziej doświadczony programista, który dba o całokształt aplikacji, jakość kodu, wybiera najlepsze rozwiązania, technologie i narzędzia, rewizja zmian proponowanych przez programistów, refaktoryzacja kodu, logika systemu.
6. Mateusz Bugowski – Programista – odpowiada za kod od strony Backendu, czyli za logikę systemu, sporządzanie diagramów.
7. Adam Żuk – Programista – zajmuje się częścią kodu, z którą interakcję prowadzi użytkownik, autor wielu nowych funkcjonalności.
8. Dawid Ciecierski – Tester – testowanie zmian i nowych funkcjonalności aplikacji na różnych etapach implementacji kodu.
9. Aleksandra Kuźniar – Tester – testowanie finalnej wersji aplikacji z punktu widzenia Klienta.

Wszystkie osoby w grupie brały aktywny i czynny udział w całym procesie tworzenia aplikacji i przygotowywania niezbędnej dokumentacji.

Diagramy działania aplikacji

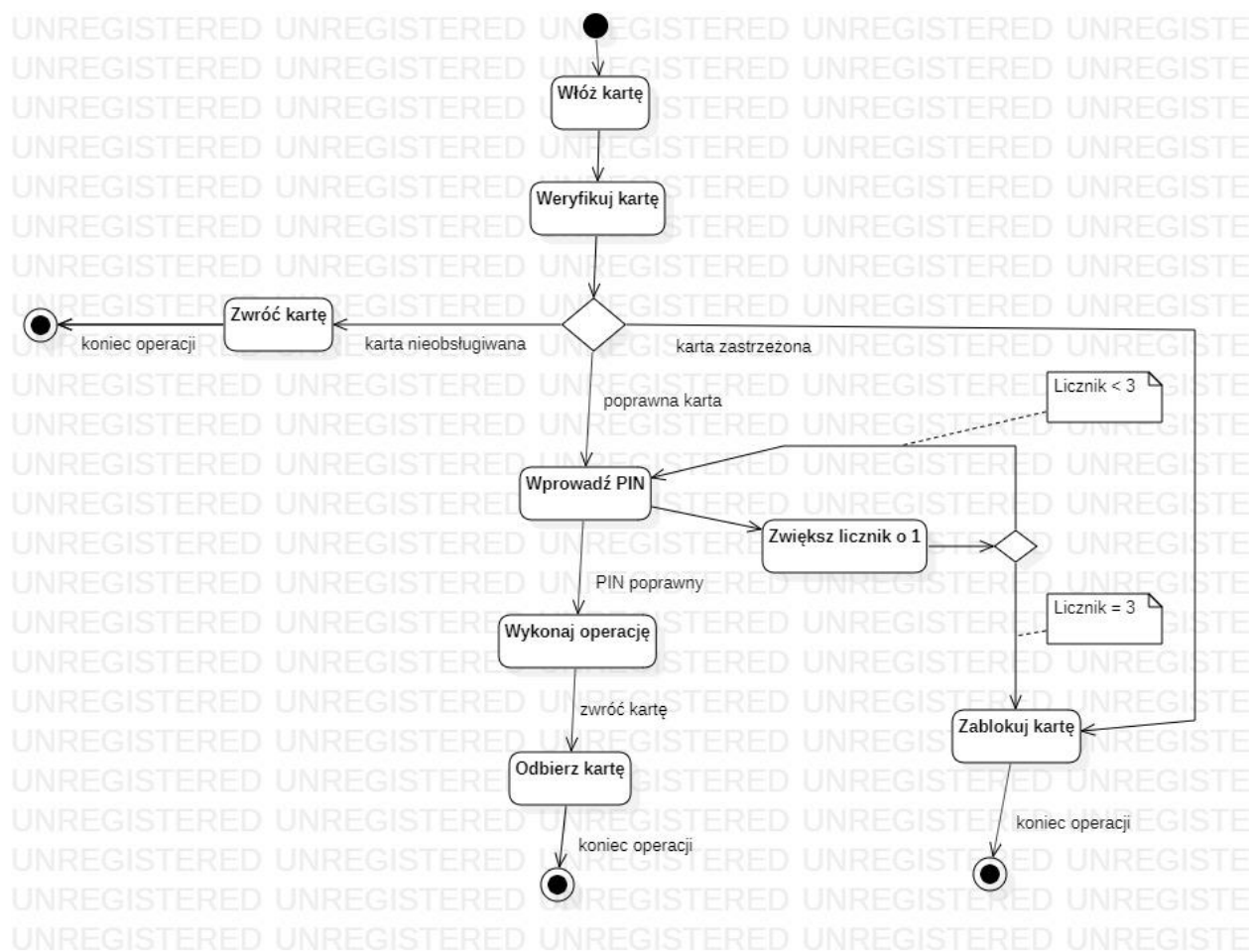
Use Case Diagram

Diagram przypadków użycia przedstawia funkcjonalność systemu wraz z jego otoczeniem. Pozwala na graficzne zaprezentowanie własności systemu tak, jak są one widziane po stronie użytkownika oraz służy do zobrazowania usług, które są widoczne z zewnątrz systemu.



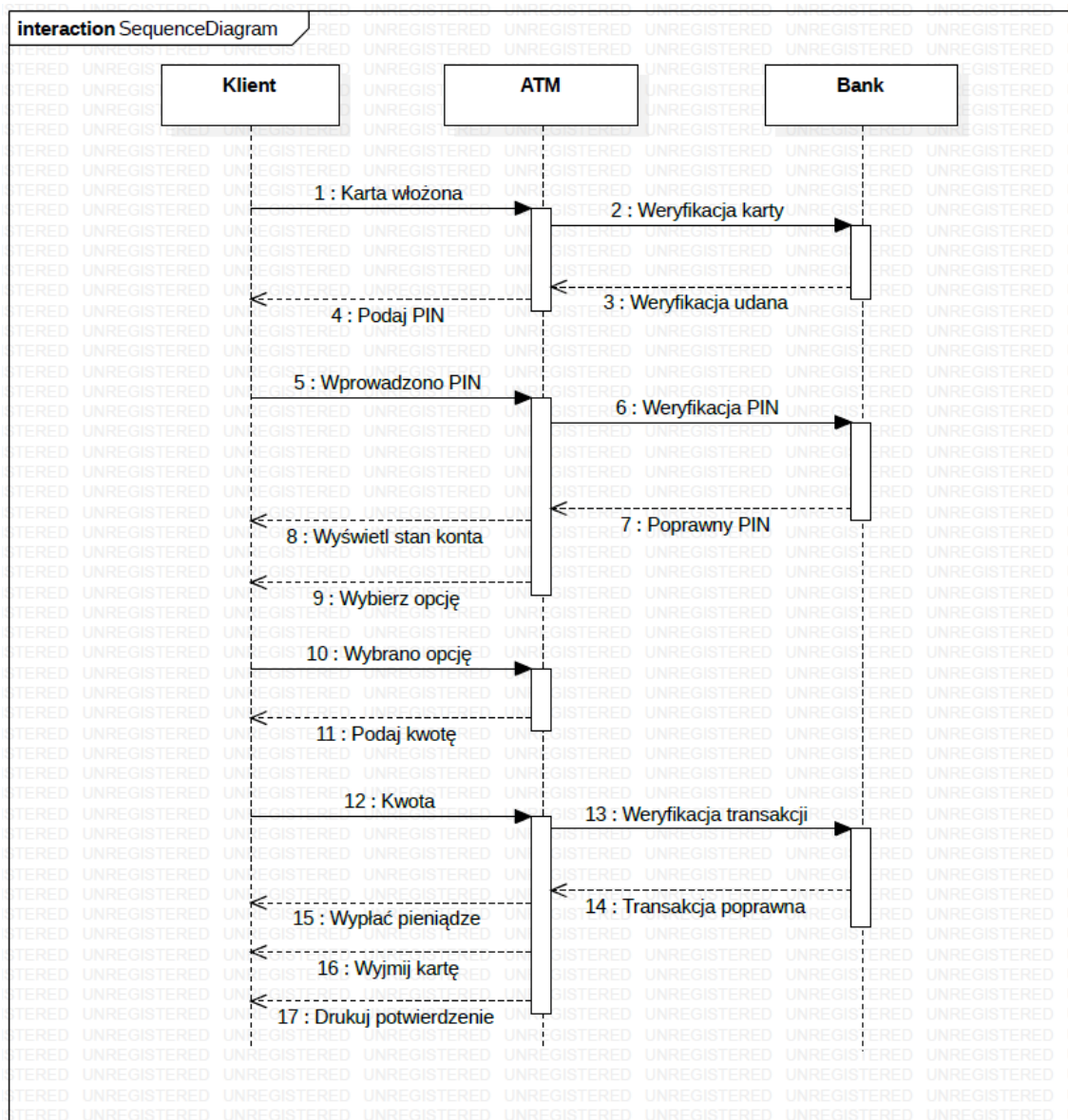
Activity Diagram

Diagram aktywności jest diagramem interakcji, który służy do modelowania dynamicznych aspektów systemu. Na diagramie widoczne są kolejne czynności, które wykonuje Klient i Bankomat, uwzględniając wszystkie możliwe przypadki, od włożenia karty do bankomatu, poprzez weryfikację i wprowadzenie kodu PIN, aż po wykonanie operacji i odbiór karty. Również opisane zostały przypadki, w których wykonanie operacji jest niemożliwe, jak włożenie nieobsługiwanej karty, karty zastrzeżonej, czy trzykrotne wpisanie błędnego PINu.



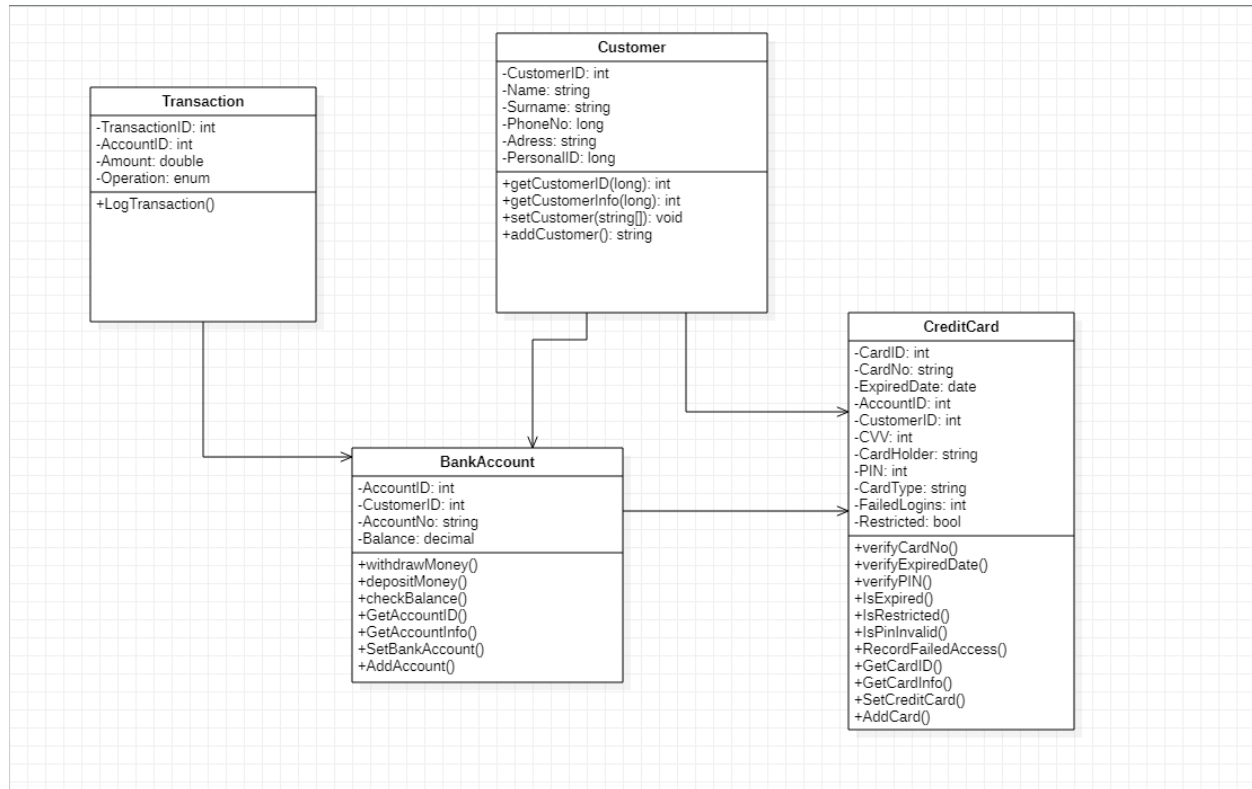
Sequence Diagram

Diagram sekwencji służy do prezentowania interakcji pomiędzy obiektami wraz z uwzględnieniem w czasie komunikatów, jakie są przesyłane pomiędzy nimi. Zasadniczym zastosowaniem diagramów sekwencji jest modelowanie zachowania systemu w kontekście scenariuszy przypadków użycia. Diagramy sekwencji pozwalają uzyskać odpowiedź na pytanie, jak w czasie przebiega komunikacja pomiędzy obiektami. Dodatkowo diagramy sekwencji stanowią podstawową technikę modelowania zachowania systemu, które składa się na realizację przypadku użycia. Poniższy diagram prezentuje poprawnie wykonaną operację wypłaty gotówki.



Class Diagram

Diagram klas przedstawia strukturę systemu w modelach obiektowych przez ilustrację struktury klas i zależności między nimi.



Podsumowanie

Z punktu widzenia Klienta projekt symuluje działanie aplikacji bankomatu w wersji podstawowej, pozwalając na wypłaty i wpłaty gotówki. Po stronie Pracownika aplikacja pozwala na obsługę kilku dodatkowych usług, jak przegląd klientów, kont i kart kredytowych, oraz wypłaty i wpłaty gotówki, symulując w ten sposób działanie okienka kasowego banku. Projekt można z powodzeniem rozbudować o funkcje i moduły do obsługi wykonywania przelewów i wyświetlania historii transakcji z wykorzystaniem dodatkowej klasy Transaction oraz wykorzystać do tego celu bazę Transactions, która została już zdefiniowana w bazie danych.