# Wymagania wstępne i projekt — Symulacja bankomatu (ATM) w C++

Cel: przygotować analizę wymagań i wstępny projekt aplikacji symulującej działanie bankomatu przy użyciu paradygmatu programowania obiektowego. Materiał zawiera analizę obiektową, propozycję architektury, opis głównych klas oraz diagram(y) klas w notacji PlantUML.

## 1. Zakres projektu

Aplikacja ma symulować podstawowe operacje bankomatu: logowanie kartą i PIN-em, wypłatę gotówki, sprawdzenie salda, wpłaty (opcjonalnie), przelewy (opcjonalnie), drukowanie potwierdzeń oraz obsługę ograniczeń (np. brak gotówki). System powinien być modularyzowany i przygotowany do rozszerzeń.

## 2. Wymagania funkcjonalne (FR)

1. FR1 — Autoryzacja: bankomat musi umożliwiać włożenie karty, wprowadzenie PINu i weryfikację danych z bazą.
2. FR2 — Wypłata gotówki: klient może wybrać kwotę (z limitem dziennym i limitem stanu konta); bankomat wydaje banknoty jeśli posiada wystarczające saldo w skrytce.
3. FR3 — Sprawdzenie salda: klient może zobaczyć bieżące saldo konta.
4. FR4 — Historia transakcji: wyświetlenie krótkiej historii ostatnich N transakcji.
5. FR5 — Wpłata (opcjonalnie): akceptacja wpłat (asynchroniczne księgowanie lub natychmiastowe w kontekście symulacji).
6. FR6 — Drukowanie potwierdzeń: opcja wydruku/paragonu (symulacja wydruku jako plik tekstowy).
7. FR7 — Obsługa błędów i wyjątków: niewłaściwy PIN, brak gotówki, błędy komunikacji z bazą.
8. FR8 — Sesja użytkownika: po X minutach bezczynności sesja wygasa i karta jest „zwracana” (sesja kończy się).

## 3. Wymagania niefunkcjonalne (NFR)

1. NFR1 — Język: C++ (zalecane: C++17 lub nowsze).
2. NFR2 — Architektura: modularna, OOP, testowalna (oddzielenie logiki od I/O konsolowego).
3. NFR3 — Łatwość rozszerzeń: dodawanie nowych typów transakcji bez modyfikacji istniejących klas (Open/Closed).
4. NFR4 — Bezpieczeństwo: PIN przechowywany w postaci hasha (symulacja), ograniczenie liczby nieudanych prób.
5. NFR5 — Dokumentacja: UML i komentarze do kluczowych modułów.
6. NFR6 — Interfejs: CLI lub prosty interfejs tekstowy; warstwa interfejsu oddzielona od logiki biznesowej.

## 4. Aktorzy i przypadki użycia (Use Cases)

**Aktorzy:** Klient, Administrator (serwis), System bankowy (zdalna baza).

Główne przypadki użycia: - Wykonaj transakcję wypłaty. - Sprawdź saldo. - Wykonaj wpłatę. - Wyświetl historię transakcji. - Zainicjuj sesję (włóż kartę, podaj PIN). - Obsłuż awarię/serwis (administrator sprawdza stan kasety z gotówką).

## 5. Analiza obiektowa — identyfikacja klas

Poniżej lista kandydatów na klasy wraz z krótkim opisem roli:

* **ATM** — główny kontroler; orkiestruje sesję użytkownika, współpracuje z urządzeniami fizycznymi i bankiem.
* **Card** — reprezentuje kartę bankową (numer, typ, status, powiązane konto).
* **Account** — reprezentuje konto bankowe (nr konta, saldo, historia transakcji, limity).
* **UserSession** — zarządza sesją zalogowanego użytkownika (stan, czas bezczynności).
* **Transaction (abstrakcyjna)** — podstawowa klasa transakcji; konkretne klasy: Withdrawal, Deposit, BalanceInquiry, Transfer.
* **CashDispenser** — fizyczny moduł wydający banknoty; ma magazyn banknotów.
* **CashStorage** — model przechowywania banknotów (ilości nominalów).
* **PINManager** — weryfikacja PINów, zliczanie prób, blokada karty.
* **BankDatabase** (interfejs) — abstrakcja dostępu do danych kont; w implementacji: InMemoryDatabase lub FileDatabase.
* **CardReader** — symulacja czytnika kart (dostarcza obiekt Card do ATM).
* **Screen** / **Keypad** / **ReceiptPrinter** — elementy I/O; oddzielone by umożliwić podmienność implementacji (np. testy).
* **Logger** — rejestr zdarzeń i operacji dla audytu.

## 6. Propozycja architektury i warstw

* Warstwa prezentacji: Screen, Keypad, ReceiptPrinter (CLI).
* Warstwa kontrolera: ATM, UserSession, PINManager.
* Warstwa domenowa: Account, Card, Transaction i jej podklasy.
* Warstwa infrastruktury: BankDatabase, CashDispenser, CashStorage, Logger.

Oddzielenie interfejsu sprzętowego (czytnik, dispenser) od logiki ułatwia testowanie i rozwój.

## 7. Opis głównych klas (atrybuty i metody)

Poniżej przedstawione są propozycje atrybutów i metod (nazwy w stylu C++ CamelCase). Niektóre metody są pomocnicze/wyjątkowe i służą tylko symulacji.

### Class: ATM

* **Atrybuty:** CardReader& cardReader, Screen& screen, Keypad& keypad, CashDispenser& dispenser, BankDatabase& db, PINManager pinManager, Logger& logger
* **Metody:** start(), handleCardInsertion(), authenticate(), showMainMenu(), processTransaction(Transaction\*), endSession()

### Class: Card

* **Atrybuty:** string cardNumber, string cardholderName, int expiryMonth, expiryYear, bool blocked, string linkedAccountId
* **Metody:** isExpired(), block(), unblock()

### Class: Account

* **Atrybuty:** string accountId, double balance, vector<Transaction> history, double dailyLimit
* **Metody:** credit(amount), debit(amount) -> bool, getBalance(), addTransaction(Transaction)

### Class: Transaction (abstract)

* **Atrybuty:** string id, string accountId, time\_t timestamp, double amount, enum Status {PENDING, COMPLETED, FAILED}
* **Metody (virtual):** execute(BankDatabase&, CashDispenser&), toString()

### Class: Withdrawal : public Transaction

* **Metody:** execute(...) — sprawdza saldo konta i dostępność banknotów, aktualizuje stan.

### Class: CashDispenser

* **Atrybuty:** CashStorage storage, int maxDispensePerTransaction
* **Metody:** canDispense(amount) -> bool, dispense(amount) -> map<nominal,int>, replenish(map<nominal,int>)

### Class: PINManager

* **Atrybuty:** map<string,int> failedAttemptsByCard, int maxAttempts
* **Metody:** verify(card, inputPin, BankDatabase&) -> bool, recordFailure(card), resetAttempts(card)

### Class: BankDatabase (interfejs/abstrakcja)

* **Metody:** getAccount(accountId) -> Account\*, updateAccount(Account\*), verifyPin(cardNumber, pinHash) -> bool, logTransaction(Transaction)

### Class: Logger

* **Metody:** logInfo(string), logError(string), logTransaction(Transaction)

## 8. Zależności i relacje (w skrócie)

* ATM używa CardReader, Screen, Keypad, PINManager, CashDispenser i BankDatabase.
* Card powiązana z Account (1:1 lub 1:n w zależności od rozszerzeń).
* Transaction powiązana z Account (agregacja) i rejestrowana przez BankDatabase.
* CashDispenser używa CashStorage do ustalania dostępności banknotów.

## 9. Diagram klas (PlantUML)

Poniżej zamieszczam kod PlantUML, który można wkleić do narzędzia renderującego (np. PlantUML, online viewer) by otrzymać diagram klas.

@startuml  
skinparam classAttributeIconSize 0  
  
class ATM {  
 - CardReader& cardReader  
 - Screen& screen  
 - Keypad& keypad  
 - CashDispenser& dispenser  
 - BankDatabase& db  
 - PINManager pinManager  
 + start()  
 + handleCardInsertion()  
 + authenticate()  
 + showMainMenu()  
 + processTransaction(t : Transaction\*)  
 + endSession()  
}  
  
class Card {  
 - string cardNumber  
 - string cardholderName  
 - int expiryMonth  
 - int expiryYear  
 - bool blocked  
 - string linkedAccountId  
 + isExpired()  
 + block()  
}  
  
class Account {  
 - string accountId  
 - double balance  
 - double dailyLimit  
 - vector<Transaction> history  
 + credit(a)  
 + debit(a) : bool  
 + getBalance()  
}  
  
abstract class Transaction {  
 - string id  
 - string accountId  
 - double amount  
 - time\_t timestamp  
 - Status status  
 + execute(db : BankDatabase, disp : CashDispenser)  
}  
  
class Withdrawal {  
 + execute(db, disp)  
}  
  
class CashDispenser {  
 - CashStorage storage  
 - int maxDispensePerTransaction  
 + canDispense(amount) : bool  
 + dispense(amount) : map  
 + replenish(map)  
}  
  
class PINManager {  
 - map<string,int> failedAttemptsByCard  
 - int maxAttempts  
 + verify(card, pin, db) : bool  
 + recordFailure(card)  
 + resetAttempts(card)  
}  
  
interface BankDatabase {  
 + getAccount(id) : Account\*  
 + updateAccount(a)  
 + verifyPin(cardNumber, pinHash) : bool  
 + logTransaction(t : Transaction)  
}  
  
class Logger {  
 + logInfo(s)  
 + logError(s)  
}  
  
ATM --> CardReader  
ATM --> Screen  
ATM --> Keypad  
ATM --> CashDispenser  
ATM --> BankDatabase  
ATM --> PINManager  
ATM --> Logger  
Card "1" -- "1" Account  
Account "1" o-- "\*" Transaction  
Transaction <|-- Withdrawal  
CashDispenser o-- CashStorage  
BankDatabase <|.. InMemoryDatabase  
  
@enduml

## 10. Przykładowy przepływ (sekwencja) dla wypłaty

1. Klient wkłada kartę (CardReader -> ATM).
2. ATM prosi o PIN (Screen -> Keypad).
3. PINManager weryfikuje PIN przez BankDatabase.
4. Po autoryzacji ATM pobiera dane konta i pokazuje menu.
5. Klient wybiera wypłatę; ATM tworzy obiekt Withdrawal.
6. Withdrawal::execute() sprawdza saldo i dostępność banknotów w CashDispenser.
7. Jeśli OK: CashDispenser::dispense() fizycznie wydaje banknoty, BankDatabase::updateAccount() zmniejsza saldo i BankDatabase::logTransaction() rejestruje transakcję.
8. ReceiptPrinter tworzy paragon, sesja kończy się lub wraca do menu.

## 11. Dalsze kroki i rozszerzenia

* Implementacja testów jednostkowych (mockowanie BankDatabase, CashDispenser).
* Dodanie warstwy komunikacji sieciowej (symulacja połączenia z centralnym serwerem banku).
* Udoskonalenie modelu CashStorage (algorytm optymalnego wydawania banknotów).
* Interfejs graficzny (opcjonalny) lub API REST do integracji z frontendem.