SISTEM PERBANKAN SEDERHANA

Rafid Farhan[1], Ariela Safmi[2], Muhammad Musa[3,]
Ahmad Muwafiqul[4], Yusuf Sohibul[5], Kanaya Dzikra[6]

Universitas Koperasi Indonesia
knydzkraa435@gmail.com [1] | arielsafmi@gmail.com [2] | yusufsf01@gmail.com [3] |
ahmadmuwafiquladli@gmail.com [4] | bankbro24@gmail.com [5] |
muhammadmusaibrahim699@gmail.com [6]

ABSTRAK

Negara Indonesia sebagai Negara yang sedang berkembang bertujuan untuk mencapai masyarakat yang adil dan makmur, merata materil dan juga spiritual. Sebagai Negara yang memiliki tujuan dan strategi, Indonesia menerapkan banyak hal yang harus dikerjakan untuk meningkatkan perekonomian Indonesia yang matang disertai dengan pemerataan pembangunan yang tersebar ke seluruh pelosok. Sistem perbankan memegang peran fundamental dalam sistem keuangan modern. Memahami cara kerjanya sangatlah penting. Membangun sistem perbankan sederhana dapat membantu dalam memahami konsep dasar dan fungsionalitasnya. Paper ini menjelaskan desain dan implementasi sistem perbankan sederhana yang mensimulasikan beberapa fungsi dasar seperti pembuatan akun, setoran, penarikan, dan pemeriksaan saldo. Sistem ini menggunakan konsep pemrograman berorientasi objek (OOP) untuk membangun kelas-kelas yang mewakili akun bank, transaksi, dan bank itu sendiri.

Kata kunci: *Perbankan, Setoran, Penarikan, Saldo*

1. PENDAHULUAN

Sistem perbankan menyediakan layanan pengelolaan keuangan bagi individu dan organisasi. Sistem ini memungkinkan nasabah untuk menyimpan dana, melakukan transfer, meminjam uang, dan melakukan berbagai transaksi keuangan lainnya. Memahami cara kerja sistem perbankan penting bagi siapa saja yang ingin mengelola keuangan mereka secara efektif.

Membangun sistem perbankan sederhana dapat membantu dalam memahami konsep dasar dan fungsionalitas sistem perbankan yang kompleks. Sistem simulasi ini memungkinkan pengguna untuk bereksperimen dengan berbagai fungsi perbankan, seperti:

- Membuat akun bank baru, dengan memasukkan nama dan saldo awal.
- Melakukan setoran, ke akun bank yang ada dengan memasukkan nomor akun dan jumlah setoran.
- Melakukan penarikan, dari akun bank yang ada dengan memasukkan nomor akun dan jumlah penarikan.
- Memeriksa saldo, akun bank yang ada dengan memasukkan nomor akun.
- Melihat riwayat transaksi, untuk akun bank yang ada.

2. TINJAUAN PUSTAKA

2. 1 Dasar Teori

a. Sistem Perbankan

Menurut Lukman Dendawijaya (2005:14), Bank adalah suatu badan usaha yang tugas utamanya sebagai lembaga perantara keuangan (financial intermediaries), yang menyelurkan dana dari pihak yang kelebihan dana (surplus unit) kepada pihak yang membutuhkan dana atau kekurangan dana (deficit unit) pada waktu yang ditentukan

b. Sistem Perbankan

Menurut Frederic S. Mishkin. (2004), Sistem perbankan adalah jaringan lembaga keuangan yang menyediakan layanan terkait dengan penyimpanan uang, penyaluran kredit, dan layanan keuangan lainnya. Menurutnya, sistem perbankan adalah kunci dalam proses intermediasi keuangan yang menghubungkan penabung dengan peminjam.

c. Lembaga Keuangan

Menurut SK Menkeu RI (No.792 Tahun 1990), Lembaga keuangan adalah semua badan yang kegiatannya bidang keuangan, melakukan penghimpunan dan penyaluran dana kepada masyarakat terutama guna membiayai investasi perusahaan.

d. Setoran

Menurut Soemarso S.R. (2004), setoran adalah uang yang diterima oleh bank dari nasabah untuk ditambahkan ke rekening nasabah yang bersangkutan. Setoran ini akan dicatat dalam akun kas bank dan akun nasabah yang bersangkutan.

e. Penarikan

Menurut Mulyadi. (2001), penarikan adalah transaksi yang dilakukan oleh nasabah untuk mengurangi saldo dalam rekeningnya di bank. Penarikan ini merupakan bagian dari aktivitas perbankan yang melibatkan interaksi antara nasabah dan bank.

f. Saldo

Menurut Hanafi dan Halim. (2009), saldo adalah jumlah uang yang tersisa dalam rekening setelah dilakukan berbagai transaksi keuangan, seperti penarikan, setoran, dan transfer. Saldo ini penting untuk mencerminkan likuiditas dan kemampuan pembayaran dari pemegang rekening.

g. Riwayat Transaksi

Menurut Kasmir. (2008), Riwayat transaksi adalah dokumentasi dari semua aktivitas keuangan yang terjadi di dalam rekening nasabah. Riwayat ini termasuk setoran, penarikan, pembayaran, dan transfer, yang dicatat oleh bank sebagai bukti transaksi.

2. 2 Konsep Pemrograman Berorientasi Objek (OOP)

a. Kelas (class)
Menurut Robert Lafore, kelas adalah struktur yang mendefinisikan tipe objek, yang menyatakan data yang akan disimpan dan operasi-operasi yang dilakukan pada data tersebut.

b. Objek (object)
Menurut Betrand Meyer, objek adalah entitas yang menyimpan data dalam bentuk variabel, dan memiliki metode untuk mengakses dan memanipulasi data tersebut.

c. Enkapsulasi
Menurut Grady Booch, enkapsulasi adalah pembungkusan bersama data dan metode yang bekerja pada data tersebut dalam satu unit Tunggal

d. Pewarisan (inheritance)
Menurut Herbert Schildt, pewarisan adalah konsep utama dalam pemrograman berbasis objek yang memungkinkan kelas untuk mewarisi metode dan variabel dari kelas lain.

e. Polimorfosis
Menurut Grady Booch, polimorfisme adalah sifat yang memungkinkan entitas (seperti fungsi atau operator) untuk mengambil berbagai bentuk atau tipe data

3. HASIL PENELITIAN DAN PEMBAHASAN

a. Input

```
#include <iostream>
#include <unordered_map>
#include <ctime>
#include <iomanip>
#include <limits>
#include <memory>
```

```
#include <string>
#include <vector>

#ifdef _WIN32
#include <cstdlib>

void clearScreen() {
    std::system("cls");
}
#else
#include <cstdlib>

void clearScreen() {
    std::system("clear");
}
#endif

// Class Transaction
class Transaction {
private:
    std::string transactionType;
    int amount;
    std::time_t transactionDate;

public:
    Transaction(const std::string& type, int amt) {
        transactionType = type;
        amount = amt;
        transactionDate = std::time(nullptr);
    }

    std::string getTransactionType() const {
        return transactionType;
    }

    int getAmount() const {
        return amount;
    }

    std::string getTransactionDate() const {
        char buffer[20];
        std::strftime(buffer,  sizeof(buffer),  "%Y-%m-%d  %H:%M:%S",
std::localtime(&transactionDate));
        return std::string(buffer);
    }
};

// Class Account
class Account {
private:
    std::string accountNumber;
    std::string accountHolder;
    int balance;
    std::vector<Transaction> transactions;

public:
    Account(const std::string& number, const std::string& holder, int
initialBalance) {
        accountNumber = number;
        accountHolder = holder;
```

```
        balance = initialBalance;
    }

    std::string getAccountNumber() const {
        return accountNumber;
    }

    std::string getAccountHolder() const {
        return accountHolder;
    }

    int getBalance() const {
        return balance;
    }

    bool deposit(int amount) {
        if (amount > 0) {
            balance += amount;
            transactions.emplace_back("Deposit", amount);
            return true;
        }
        return false;
    }

    bool withdraw(int amount) {
        if (balance >= amount && amount > 0) {
            balance -= amount;
            transactions.emplace_back("Withdrawal", amount);
            return true;
        }
        return false;
    }

    void printTransactions() const {
        std::cout << std::setw(25) << std::left << "Transaction Type"
                  << std::setw(25) << std::left << "Amount"
                  <<  std::setw(25)  <<  std::left  <<  "Date"  <<
std::endl;
        std::cout << std::string(75, '-') << std::endl;
        for (const Transaction& transaction : transactions) {
            std::cout    <<    std::setw(25)    <<    std::left    <<
transaction.getTransactionType()
                      <<  "Rp  "  <<  std::setw(25)  <<  std::left  <<
transaction.getAmount()
                      <<     std::setw(25)     <<     std::left     <<
transaction.getTransactionDate() << std::endl;
        }
    }
};

// Class Bank
class Bank {
private:
    std::unordered_map<std::string,         std::unique_ptr<Account>>
accounts;

public:
    void addAccount(std::unique_ptr<Account> account) {
        std::string accountNumber = account->getAccountNumber();
        accounts[accountNumber] = std::move(account);
```

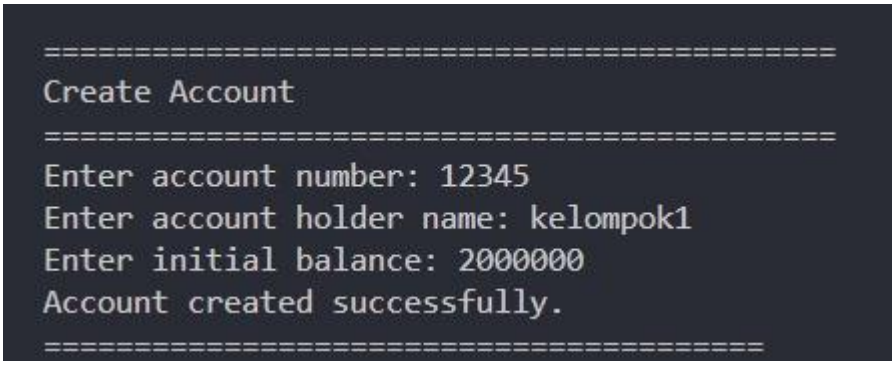
```
    }

    bool accountExists(const std::string& accountNumber) const {
        return accounts.find(accountNumber) != accounts.end();
    }

    bool deposit(const std::string& accountNumber, int amount) {
        auto it = accounts.find(accountNumber);
        if (it != accounts.end()) {
            return it->second->deposit(amount);
        }
        return false;
    }

    bool withdraw(const std::string& accountNumber, int amount) {
        auto it = accounts.find(accountNumber);
        if (it != accounts.end()) {
            return it->second->withdraw(amount);
        }
        return false;
    }

    int getBalance(const std::string& accountNumber) const {
        auto it = accounts.find(accountNumber);
        if (it != accounts.end()) {
            return it->second->getBalance();
        }
        return 0;
    }

    void printTransactions(const std::string& accountNumber) const {
        auto it = accounts.find(accountNumber);
        if (it != accounts.end()) {
            std::cout << "Transaction history for account " <<
accountNumber << ":" << std::endl;
            it->second->printTransactions();
        } else {
            std::cout << "Account " << accountNumber << " does not
exist." << std::endl;
        }
    }
};

// Function to display ATM menu
void displayMenu() {
    std::cout << "========================================" <<
std::endl;
    std::cout << std::setw(40) << std::left << "Welcome to ATM" <<
std::endl;
    std::cout << "========================================" <<
std::endl;
    std::cout << std::setw(40) << std::left << "1. Create Account" <<
std::endl;
    std:: cout << std::setw(40) << std::left << "2. Deposit" <<
std::endl;
    std::cout << std::setw(40) << std::left << "3. Withdrawal" <<
std::endl;
    std::cout << std::setw(40) << std::left << "4. Check Balance" <<
std::endl;
```

```
    std::cout << std::setw(40) << std::left << "5. Transaction History"
<< std::endl;
    std::cout << std::setw(40) << std::left << "6. Exit" << std::endl;
    std::cout  <<  "========================================="  <<
std::endl;
    std::cout << "Enter your choice: ";
}

// Function to pause execution (for Windows only)
void pause() {
    #ifdef _WIN32
        system("pause");
    #endif
}

// Main function
int main() {
    Bank bank;

    int choice;
    std::string accountNumber;
    std::string accountHolder;
    int amount;

    do {
        clearScreen();
        displayMenu();

        if (!(std::cin >> choice)) {
            std::cout  <<  "Invalid  input.  Please  try  again."  <<
std::endl;
            std::cin.clear();

std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            pause(); // Pause execution after displaying error message
            continue;
        }

        switch (choice) {
            case 1:
                system("cls");
                std::cout                                              <<
"=========================================" << std::endl;
                std::cout << "Create Account" << std::endl;
                std::cout                                              <<
"=========================================" << std::endl;
                std::cout << "Enter account number: ";
                std::cin >> accountNumber;

                if (bank.accountExists(accountNumber)) {
                    std::cout << "Account number already exists. Try
again with a different number." << std::endl;
                    break;
                }

                std::cout << "Enter account holder name: ";
                std::cin.ignore();  //  Ignore  the  newline  character
left in the buffer
                std::getline(std::cin, accountHolder);
                std::cout << "Enter initial balance: ";
```

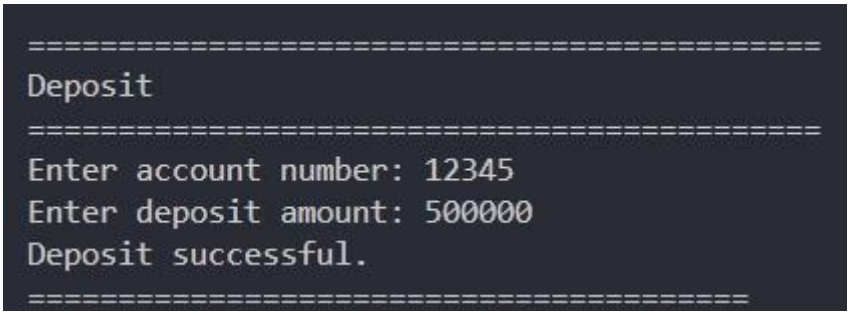
```
                if (!(std::cin >> amount) || amount < 0) {
                    std::cout << "Invalid initial balance." <<
std::endl;
                    system("pause");
                    break;
                }

bank.addAccount(std::make_unique<Account>(accountNumber,
accountHolder, amount));
                std::cout << "Account created successfully." <<
std::endl;

                //system("pause");
                break;

            case 2:
                system("cls");
                std::cout                                          <<
"========================================" << std::endl;
                std::cout << "Deposit" << std::endl;
                std::cout                                          <<
"========================================" << std::endl;
                std::cout << "Enter account number: ";
                std::cin >> accountNumber;
                std::cout << "Enter deposit amount: ";
                if (!(std::cin >> amount) || amount <= 0) {
                    std::cout << "Invalid deposit amount." <<
std::endl;
                    system("pause");
                    break;
                }
                //system("cls");
                if (bank.deposit(accountNumber, amount)) {
                    std::cout << "Deposit successful." << std::endl;
                } else {
                    std::cout << "Invalid account number." <<
std::endl;
                }
                //system("pause");
                break;

            case 3:
            system("cls");
                std::cout                                          <<
"========================================" << std::endl;
                std::cout << "Withdrawal" << std::endl;
                std::cout                                          <<
"========================================" << std::endl;
                std::cout << "Enter account number: ";
                std::cin >> accountNumber;
                std::cout << "Enter withdrawal amount: ";
                if (!(std::cin >> amount) || amount <= 0) {
                    std::cout << "Invalid withdrawal amount." <<
std::endl;
                    break;
                }

                if (bank.withdraw(accountNumber, amount)) {
                    std::cout << "Withdrawal successful." <<
std::endl;
```

```
                } else {
                    std::cout << "Invalid account number or
insufficient balance." << std::endl;
                }
                system("pause");
                break;

            case 4:
            system("cls");
                std::cout                                          <<
"========================================" << std::endl;
                std::cout << "Check Balance" << std::endl;
                std::cout                                          <<
"========================================" << std::endl;
                std::cout << "Enter account number: ";
                std::cin >> accountNumber;
                std::cout << "Balance for account " << accountNumber
<< ": Rp " << bank.getBalance(accountNumber) << std::endl;
                system("pause");
                break;

            case 5:
            system("cls");
                std::cout                                          <<
"========================================" << std::endl;
                std::cout << "Transaction History" << std::endl;
                std::cout                                          <<
"========================================" << std::endl;
                std::cout << "Enter account number: ";
                std::cin >> accountNumber;
                bank.printTransactions(accountNumber);
                system("pause");
                break;

            case 6:
                std::cout << "Exiting..." << std::endl;
                break;

            default:
                std::cout << "Invalid choice. Try again." <<
std::endl;
                break;
        }

        std::cout << std::string(40, '=') << std::endl; // Print
separator after each menu
        pause(); // Pause execution after displaying menu
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');

    } while (choice != 6);

    return 0;
}
```

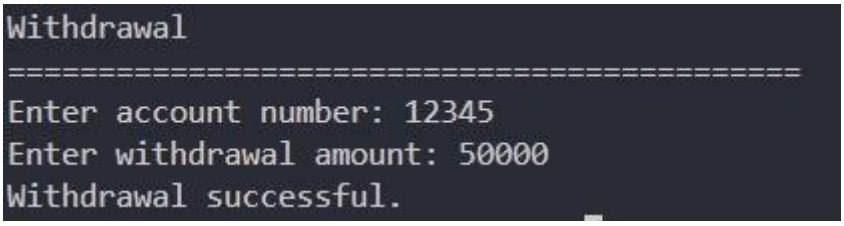
b. Output



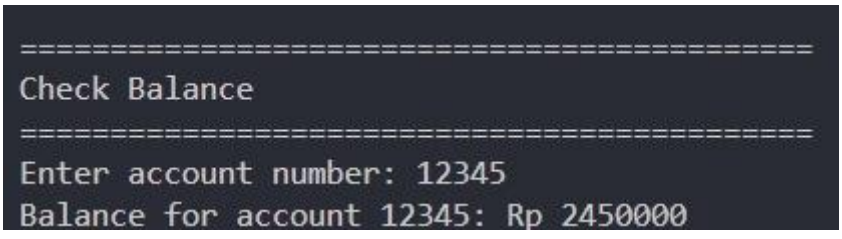
Gambar 1 Create Account / Membuat Akun



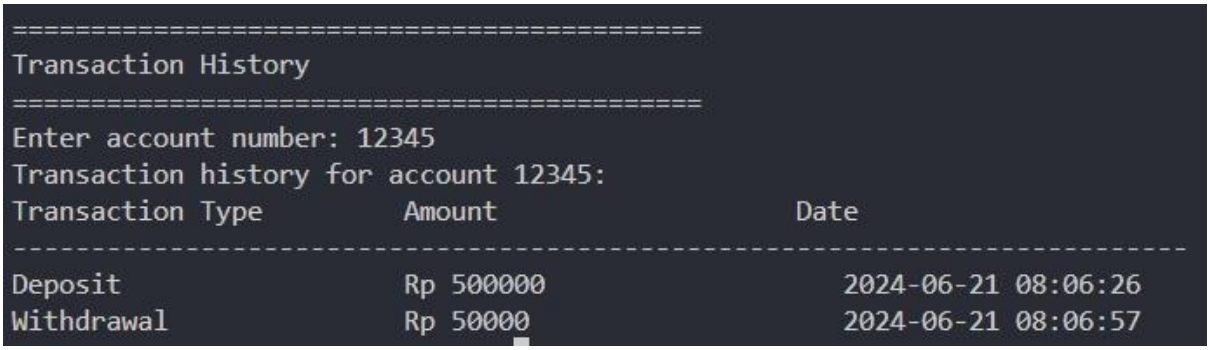
Gambar 2 Deposit



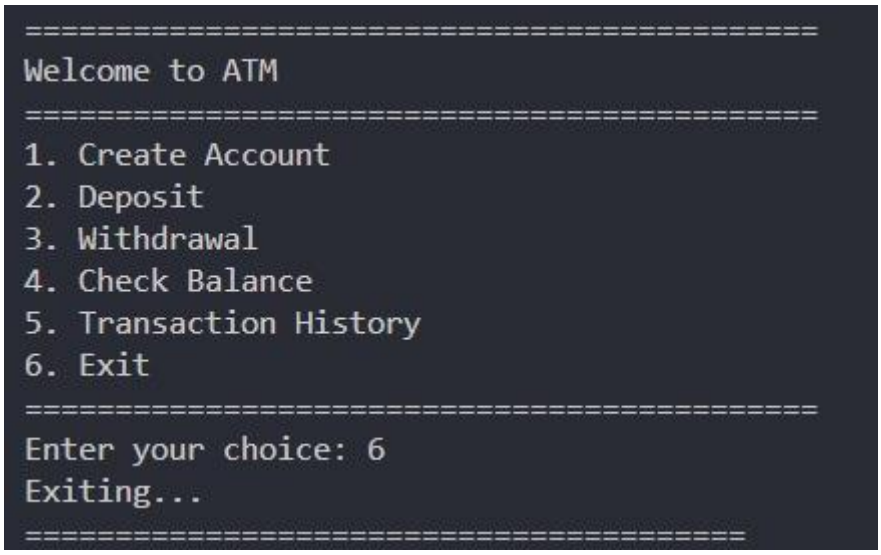
Gambar 3 Withdrawal / Penarikan



Gambar 4 Check Balance / Cek Saldo



Gambar 5 Transaction History / Riwayat Transaksi



Gambar 6 Exit / Keluar

4. KESIMPULAN

Sistem perbankan sederhana ini memberikan simulasi fungsionalitas dasar dari sistem perbankan yang kompleks. Sistem ini dibangun menggunakan konsep OOP dan menunjukkan bagaimana kelas, inheritance, polymorphism, exception handling, dan encapsulation dapat digunakan untuk membangun aplikasi yang robust dan skalabel. Implementasi sistem ini dapat membantu dalam memahami konsep-konsep penting dalam pemrograman dan sistem perbankan.

DAFTAR PUSTAKA

Simatupang. H Bachtiar. (2019). *Peranan Perbankan dalam Meningkatkan Perekonimian Indonesia*. Sumatera Utara: Universitas Islam Sumatera Utara

Hermalia Putri. (2023). *Program Sistem Bank Sederhana menggunakan C++.* Bandung: Medium.com