Sorana Ioana Vicu

Documenație pentru rezolvarea problemelor pentru disciplina Complemente de programare

1. Documentație detaliată pentru aplicația de gestionare a contului bancar C++

Introducere

Această documentație descrie dezvoltarea și funcționarea unei aplicații C++ care gestionează un cont bancar. Aplicația permite utilizatorului să efectueze tranzacții recurente, să aplice dobânda lunară și să verifice dacă limita de credit este depășită. Aplicația este interactivă și folosește intrări de la utilizator prin intermediul consolei.

Structura Codului

Proiectul constă într-un singur fișier C++ care conține definiția structurii `ACCOUNT`, funcțiile necesare pentru gestionarea contului și programul principal care interacționează cu utilizatorul.

main.cpp

```cpp
#include <iostream>
using namespace std;

// Declararea structurii pentru contul bancar
struct ACCOUNT {
    double balance;
    double interestRate;
    double interest;
    double creditLimit;
};

// Funcția pentru inițializarea contului
void initializeAccount(ACCOUNT &account, double initialBalance, double initialInterestRate, double creditLimit)
{
    account.balance = initialBalance;
    account.interestRate = initialInterestRate;
    account.creditLimit = creditLimit;
    account.interest = 0;
}

// Funcție pentru efectuarea unei plăți recurente care include depășirea limitei de credit
bool makeTransaction(ACCOUNT &account, double paymentAmount) {
    if (account.balance - paymentAmount < -account.creditLimit) {
        cout << "Limita de credit depășită" << endl;
        return false;
    }
    account.balance -= paymentAmount;
    return true;
}
```

Sorana Ioana Vicu

```cpp
// Funcție pentru aplicarea dobânzii lunare
void applyInterest(ACCOUNT &account) {
    account.interest = account.balance * account.interestRate / 100;
    if (account.interest > 0) {
        account.balance += account.interest;
    } else {
        cout << "Dobânda nu poate fi aplicată din lipsă de fonduri." << endl;
    }
}

// Programul principal
int main() {
    ACCOUNT SoranaVicu;
    double initialBalance, interestRate, creditLimit;
    double paymentAmount, priceTag;
    int months;

    // Inițializarea contului plus o condiție de reintroducere a datelor în caz că au fost introduse greșit
    int retryTransaction = 1;
    cout << "Acesta este o aplicație care vă ajută să vă calculați viitoarele rate pentru posibile cumpărături." << endl;
    cout << " " << endl;
    cout << " " << endl;
    cout << "Introduceți valoarea inițială din contul dumneavoastră: ";
    cin >> initialBalance;
    cout << "Introduceți rata dobânzii lunare (% în procente): ";
    cin >> interestRate;
    cout << "Introduceți limita de credit: ";
    cin >> creditLimit;
    cout << " " << endl;
```

```cpp
    while (retryTransaction == 1) {
        initializeAccount(SoranaVicu, initialBalance, interestRate, creditLimit);


        // Setare plăți recurente
        cout << " " << endl;
        cout << "Introduceți costul cumpărăturii: ";
        cin >> priceTag;
        cout << "Introduceți numărul de tranșe: ";
        cin >> months;
        paymentAmount = priceTag / months;
        cout << "Valoarea ratelor lunare este de " << paymentAmount << " LEI" << endl;
        cout << " " << endl;


        // Simulare pentru fiecare lună
        for (int i = 1; i <= months; ++i) {
            cout << "Luna " << i << ":" << endl;
            if (!makeTransaction(SoranaVicu, paymentAmount)) {
                cout << "Plata de " << paymentAmount << " LEI nu poate fi efectuată. " << endl;
                retryTransaction = 0;
                break;
            }
            retryTransaction = 1;
            cout << "Plata de " << paymentAmount << " LEI a fost efectuată cu succes!" << endl;
            cout << "Sold nou: " << SoranaVicu.balance << " LEI" << endl;

            applyInterest(SoranaVicu);
            if (SoranaVicu.interest > 0) {
                cout << "Dobânda a fost adăugată în contul dumneavoastră" << endl;
            }
            cout << "Soldul la sfârșitul lunii " << i << " este de: " << SoranaVicu.balance << " LEI" << endl;
            cout << " " << endl;
        }

```

```
95
96          // Afișare sold final
97          cout << " " << endl;
98          cout << " " << endl;
99
100         if (retryTransaction == 0) {
101             cout << "Soldul final al contului este: " << SoranaVicu.balance << " LEI" << endl;
102             cout << "Nu vă permiteţi acest produs!" << endl;
103         } else {
104             cout << "Soldul final al contului este: " << SoranaVicu.balance << " LEI" << endl;
105             cout << "Vă permiteţi acest produs. Spor la cumpărături!" << endl;
106         }
107
108         cout << "Doriţi să calculaţi ratele pentru alt produs? (1-DA / 0-NU)" << endl;
109         cin >> retryTransaction;
110
111
112     }
113     cout << " " << endl;
114     cout << "Vă mulţumim!" << endl;
115     return 0;
116 }
117
```

Explicația Codului

Structura ACCOUNT

Structura ACCOUNT definește contul bancar cu următoarele câmpuri:

balance: Soldul curent al contului.
interestRate: Rata dobânzii lunare (în procente).
interest: Dobânda acumulată în cont.
creditLimit: Limita de credit a contului.

Funcția initializeAccount

Această funcție inițializează contul cu valorile inițiale primite ca parametri.

Cod:

```
void initializeAccount(ACCOUNT &account, double initialBalance, double initialInterestRate,
double creditLimit) {
account.balance = initialBalance;
account.interestRate = initialInterestRate;
account.creditLimit = creditLimit;
account.interest = 0;
```

Funcția makeTransaction

Această funcție efectuează o tranzacție și verifică dacă soldul după tranzacție depășește limita de credit.

Sorana Ioana Vicu

Cod:

```
bool makeTransaction(ACCOUNT &account, double paymentAmount) {
        if (account.balance - paymentAmount < -account.creditLimit) {
                cout << "Limita de credit depășită" << endl;
                return false; }
        account.balance -= paymentAmount;
                return true;
```

Funcția applyInterest

Această funcție aplică dobânda lunară la soldul contului dacă există fonduri suficiente.

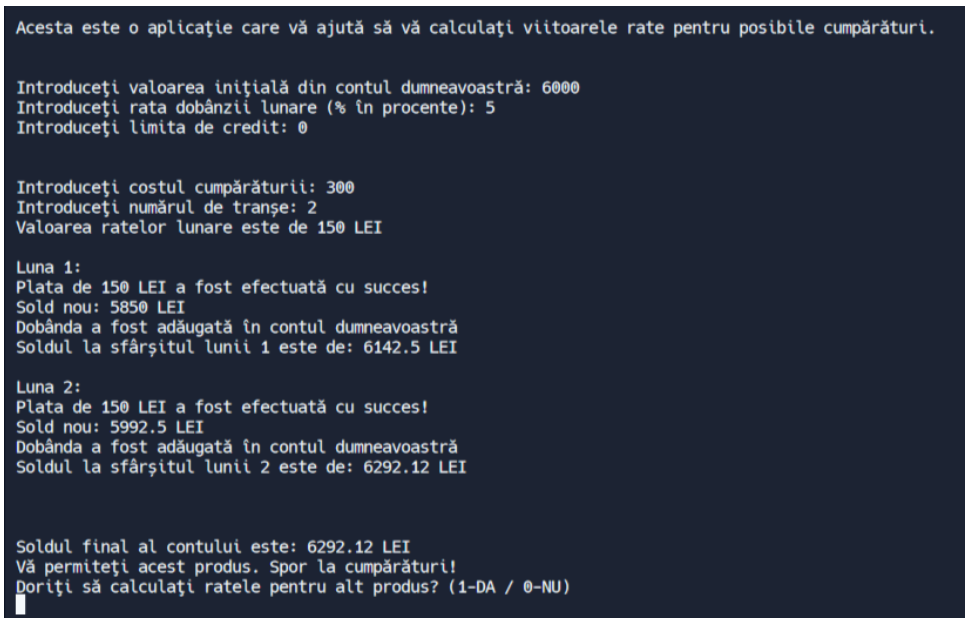
Cod:

```
void applyInterest(ACCOUNT &account) {
        account.interest = account.balance * account.interestRate / 100;
        if (account.interest > 0) {
                account.balance += account.interest;
        } else {
                cout << "Dobânda nu poate fi aplicată din lipsă de fonduri." << endl; } }
```

Programul principal

Programul principal inițializează contul, acceptă intrări de la utilizator pentru costul cumpărăturii și numărul de tranșe, simulând plățile recurente și aplicarea dobânzii pentru fiecare lună. Se regăsește de la linia 41 la linia 115 de cod.

Rularea codului



Sorana Ioana Vicu

Concluzie

Aplicația de gestionare a contului bancar dezvoltată în cadrul acestui proiect demonstrează o implementare simplă, dar eficientă, pentru simularea operațiunilor bancare de bază. Utilizatorul poate introduce datele inițiale pentru cont (soldul inițial, rata dobânzii și limita de credit), iar aplicația efectuează simularea plăților recurente pentru un anumit număr de luni, aplicând dobânda lunară și verificând depășirea limitei de credit.
Principalele funcționalități implementate includ: inițializarea contului (`initializeAccount`), efectuarea plăților (`makeTransaction`) cu gestionarea limitelor de credit și aplicarea dobânzii lunare (`applyInterest`). Programul interacționează cu utilizatorul prin intermediul consolei, afișând mesaje corespunzătoare pentru fiecare etapă a simulării.
Dezvoltarea ulterioară ar putea include îmbunătățiri ale interfeței utilizatorului, adăugarea unor opțiuni suplimentare pentru gestionarea contului și extinderea funcționalităților de simulare.

2. Documentație detaliată pentru aplicația Java BankAccount

Introducere

Această documentație descrie dezvoltarea și funcționarea unei aplicații Java care gestionează un cont bancar. Aplicația permite utilizatorului să efectueze plăți, să vizualizeze soldul curent și să verifice suma totală a plăților efectuate. Aplicația este interactivă și folosește intrări de la utilizator prin intermediul consolei.

Structura Proiectului

Proiectul constă în două fișiere principale:
`BankAccount.java`
`Main.java`

Fișierul `BankAccount.java`

Acest fișier definește clasa `BankAccount`, care gestionează operațiile bancare de bază.

Sorana Ioana Vicu

BankAccount.java | Main.java

src > main > BankAccount.java

```java
public class BankAccount {
    private double balance;
    private double totalPayments;

    // Constructor pentru inițializarea soldului inițial
    public BankAccount(double initialBalance) {
        this.balance = initialBalance;
        this.totalPayments = 0;
    }

    // Metoda pentru efectuarea plății
    public void makePayment(double amount) {
        if (this.balance >= amount) {
            this.balance -= amount;
            this.totalPayments += amount;
            System.out.println("Plata de " + amount + " este efectuată cu succes.");
        } else {
            System.out.println("Fonduri insuficiente pentru această plată.");
        }
    }

    // Metoda pentru returnarea soldului curent
    public double getBalance() {
        return this.balance;
    }

    // Metoda pentru returnarea sumei totale a plăților efectuate
    public double getTotalPayments() {
        return this.totalPayments;
    }
}
```

Explicație

Atributele Clasei:

balance: Reține soldul curent al contului.

totalPayments: Reține suma totală a plăților efectuate din cont.

Constructor:

Inițializează soldul (balance) cu valoarea primită și setează totalPayments la 0.

Metode:

makePayment(double amount): Efectuează o plată dacă soldul este suficient, actualizând balance și totalPayments.

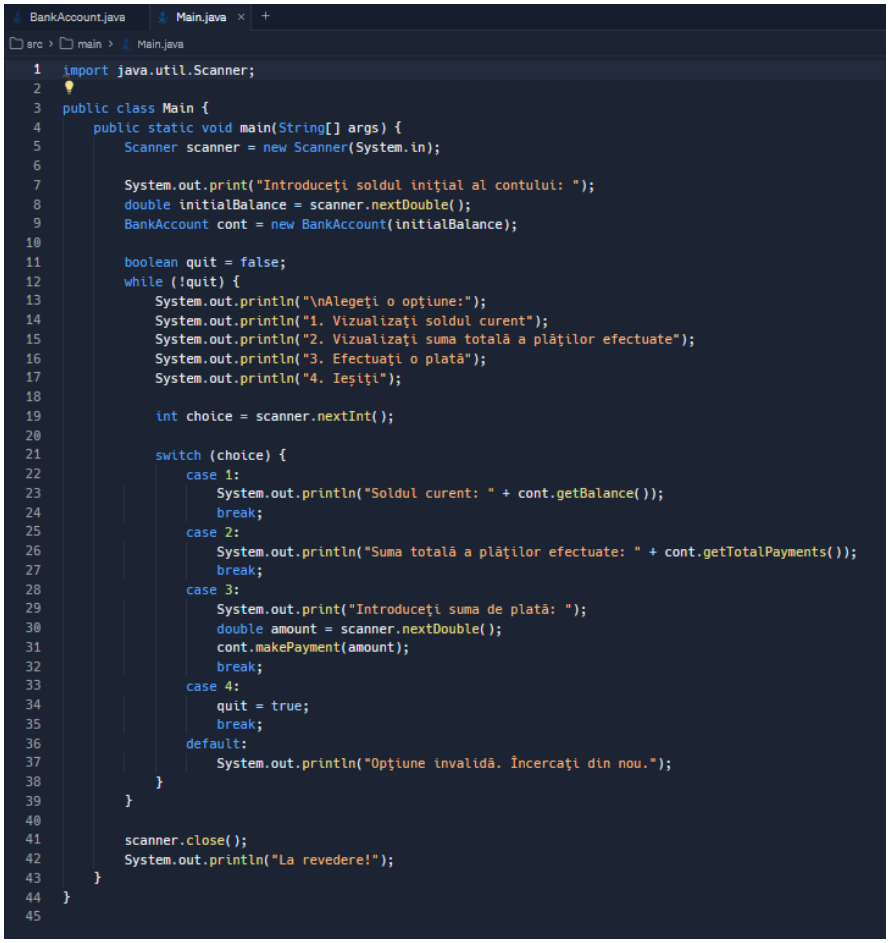
getBalance() : Returnează soldul curent.

getTotalPayments() : Returnează suma totală a plăților efectuate.

Sorana Ioana Vicu

Fișierul Main.java

Acest fișier conține logica principală a aplicației și permite interacțiunea utilizatorului cu obiectul BankAccount.



```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Introduceţi soldul iniţial al contului: ");
        double initialBalance = scanner.nextDouble();
        BankAccount cont = new BankAccount(initialBalance);

        boolean quit = false;
        while (!quit) {
            System.out.println("\nAlegeţi o opţiune:");
            System.out.println("1. Vizualizaţi soldul curent");
            System.out.println("2. Vizualizaţi suma totală a plăţilor efectuate");
            System.out.println("3. Efectuaţi o plată");
            System.out.println("4. Ieşiţi");

            int choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    System.out.println("Soldul curent: " + cont.getBalance());
                    break;
                case 2:
                    System.out.println("Suma totală a plăţilor efectuate: " + cont.getTotalPayments());
                    break;
                case 3:
                    System.out.print("Introduceţi suma de plată: ");
                    double amount = scanner.nextDouble();
                    cont.makePayment(amount);
                    break;
                case 4:
                    quit = true;
                    break;
                default:
                    System.out.println("Opţiune invalidă. Încercaţi din nou.");
            }
        }

        scanner.close();
        System.out.println("La revedere!");
    }
}
```

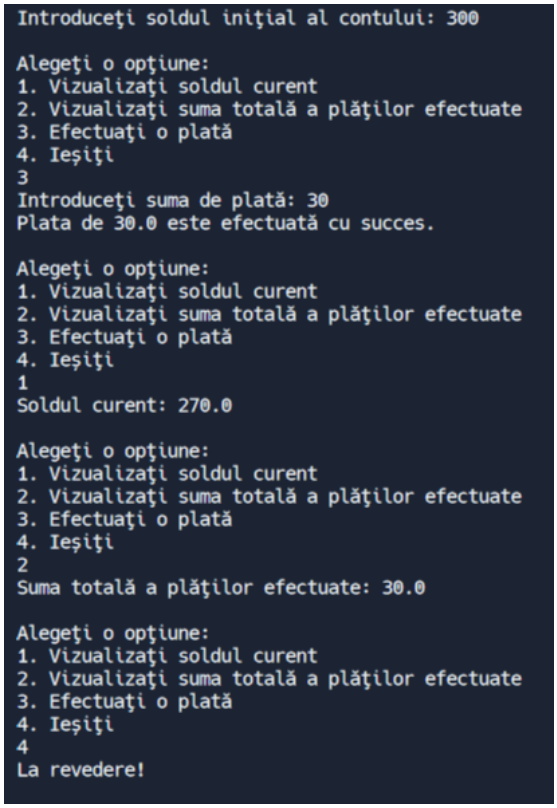
Explicație

Scanner:
Comanda Scanner este folosit pentru a citi intrările de la utilizator.

Main:
Inițializează contul bancar cu un sold inițial citit de la utilizator.
Afișează un meniu interactiv în buclă, permițând utilizatorului să aleagă între vizualizarea soldului, vizualizarea plăților totale, efectuarea unei plăți și ieșirea din aplicație.
Utilizatorul poate introduce o sumă pentru a efectua o plată, iar aplicația verifică dacă soldul este suficient înainte de a face plata.

Sorana Ioana Vicu

Rularea codului



Concluzie

Această aplicație demonstrează cum se poate gestiona un cont bancar folosind Java. Utilizatorul poate efectua plăți, verifica soldul curent și suma totală a plăților efectuate, într-un mod interactiv.