Ministerul Educației și Cercetării al Republicii Moldova

IP Centrul de Excelență în Informatică și Tehnologii Informaționale

**RAPORTUL**

**STAGIULUI DE PRACTICĂ**

Elevul:  **Axenti Dragomir**

Grupa: **W-2223**

Specialitatea: **Administrarea Aplicatiilor Web**

Baza de practică: **Centrul de excelență în Informatică și TI**

Conducătorul stagiului de practică

**MUSTEAȚĂ Victoria**

Chişinău **2024**

**Cuprins**

[Introducere 2](#_Toc169159342)

[Conținutul activităților și sarcinilor de lucru (C++) 2](#_Toc169159343)

[Formularea problemei 2](#_Toc169159344)

[Descrierea modului de elaborare a aplicației 2](#_Toc169159345)

[Codul sursă a programelor 3](#_Toc169159346)

[Rezultatele testării 3](#_Toc169159347)

[Date de intrare 3](#_Toc169159348)

[Testarea funcționalității 3](#_Toc169159349)

[Conținutul activităților și sarcinilor de lucru (Java) 3](#_Toc169159350)

[Formularea problemei 3](#_Toc169159351)

[Descrierea modului de elaborare a aplicației 4](#_Toc169159352)

[Proiectarea UML a sistemului de management 4](#_Toc169159353)

[Descrierea claselor și interfețelor 4](#_Toc169159354)

[Rezultatele testării 4](#_Toc169159355)

[Date de intrare 4](#_Toc169159356)

[Testarea funcționalității 5](#_Toc169159357)

[Concluzii 5](#_Toc169159358)

[Bibliografie 5](#_Toc169159359)

# Introducere

În cadrul stagierii de practică vom fi provocați să completăm două task-uri în limbaje de programare diferite, C++ și Java, extrem de diferite când vine vorba de abordarea pe care o vom avea față de ele. Toate cunoștințele acumulate de-a lungul anului ne vor fi testate în aceste trei săptămâni, din moment ce sarcinile încapsulează toate tehnicile de programare, modalități de a relaționa clase, a încapsula date și a implementa diverse algoritmi și structuri de date. În plus, vom fi nevoiți să demonstrăm competențe în gestionarea memoriei și optimizarea codului în C++, precum și utilizarea eficientă a bibliotecilor standard și a colecțiilor Java. Aceste provocări ne vor permite să ne perfecționăm abilitățile de problem-solving și să ne pregătim pentru proiectele complexe din viitor.

# Conținutul activităților și sarcinilor de lucru (C++)

## 

### Plan de acțiuni pentru elaborarea aplicației

#### Pasul 1: Înțelegerea și analizarea cerințelor

* **Obiectiv:** Înțelegerea clară a cerințelor aplicației.
* **Acțiuni:**
  + Studiez cerințele specificate pentru aplicație.
  + Identific funcționalitățile necesare (citirea matricei, adăugarea de linii/coloane, ștergerea elementului maxim, etc.).
  + Notific toate cerințele și întrebările pentru clarificări suplimentare dacă este necesar.

#### Pasul 2: Proiectarea soluției

* **Obiectiv:** Proiectarea structurii generale a aplicației.
* **Acțiuni:**
  + Desenez o diagramă cu structura principală a aplicației, incluzând modul în care funcțiile vor interacționa.
  + Decid asupra structurilor de date necesare (de exemplu, un tablou bidimensional pentru matrice).
  + Stabilesc interfața utilizatorului (de exemplu, meniul principal și opțiunile disponibile).

#### Pasul 3: Implementarea funcțiilor de bază

* **Obiectiv:** Implementarea funcțiilor principale ale aplicației.
* **Acțiuni:**
  + **Citirea fișierului:** Implementez funcția pentru citirea matricei dintr-un fișier.
  + **Afișarea matricei:** Implementez funcția pentru afișarea matricei în consolă.
  + **Adăugarea unei linii/coloane:** Implementez funcția pentru adăugarea unei noi linii sau coloane în matrice.
  + **Ștergerea elementului maxim:** Implementez funcția pentru ștergerea liniei și coloanei care conțin primul element maxim.

#### Pasul 4: Implementarea funcțiilor suplimentare

* **Obiectiv:** Adăugarea funcționalităților avansate.
* **Acțiuni:**
  + **Determinarea frecvenței elementelor:** Implementez funcția pentru determinarea elementului care apare cel mai frecvent în matrice.
  + **Calcularea mediei aritmetice:** Implementez funcția pentru calcularea mediei aritmetice a elementelor dintr-un interval specificat.
  + **Sortarea elementelor:** Implementez funcția pentru sortarea elementelor fiecărei linii.
  + **Găsirea subsirurilor crescătoare:** Implementez funcția pentru găsirea și scrierea subsirurilor crescătoare de lungime maximă.
  + **Parcurgerea matricei în spirală:** Implementez funcția pentru parcurgerea matricei în spirală și verificarea progresiei aritmetice.
  + **Găsirea submatricei pătratice:** Implementez funcția pentru găsirea submatricei pătratice simetrice de dimensiune maximă.

#### Pasul 5: Testarea și depanarea

* **Obiectiv:** Asigurarea funcționării corecte a aplicației.
* **Acțiuni:**
  + Testez fiecare funcție individual pentru a mă asigura că funcționează corect.
  + Efectuez teste de integrare pentru a verifica interacțiunea între diferite funcții.
  + Depanez eventualele erori și asigur stabilitatea aplicației.

#### Pasul 7: Optimizarea și documentarea

* **Obiectiv:** Optimizarea codului și crearea documentației.
* **Acțiuni:**
  + Revizuiesc și optimizez codul pentru a îmbunătăți performanța și lizibilitatea.
  + Adaug comentarii explicative în cod.

## Codul sursă a programelor

#include <bits/stdc++.h>

using namespace std;

int A[101][101];

int n,m,MX,MN;

ifstream in("Select.txt");

ofstream out("Select.txt");

void CitesteFisier(){

ifstream citeste("Matrice.in");

citeste>>n>>m;

for(int i=1;i<=n;i++)

for(int j=1;j<=m;j++){

citeste>>A[i][j];

if(A[i][j]>MX)

MX=A[i][j];

if(A[i][j]<MN)

MN=A[i][j];

}

}

void adauga(int (&a)[][101],int &n,int &m){

int x,optiune;

cout<<"Ce vreti sa introduceti?"<<endl;

cout<<"1. Linie"<<endl;

cout<<"2. Coloana"<<endl;

cin>>optiune;

if(optiune==1){

++n;

cout<<"Introduceti elementele: ";

for(int i=1;i<=m;i++){

cin>>a[n][i];

if(a[n][i]>MX)

MX=a[n][i];

else if(a[n][i]<MN)

MN=a[n][i];

}

}

if(optiune==2){

++m;

cout<<"Introduceti elementele: ";

for(int i=1;i<=n;i++){

cin>>a[i][m];

if(a[i][m]>MX)

MX=a[i][m];

else if(a[i][m]<MN)

MN=a[i][m];

}

}

ofstream fout("Matrice.in");

fout<<n<<' '<<m<<endl;

for(int i=1;i<=n;i++){

for(int j=1;j<=m;j++)

fout<<a[i][j]<<' ';

fout<<endl;

}

fout.close();

}

void afisare(int a[][101],int n, int m){

for(int i=1;i<=n;i++){

for(int j=1;j<=m;j++)

cout<<a[i][j]<<' ';

cout<<endl;

}

}

void StergeMaxim(int (&a)[101][101], int &n, int &m) {

cout<<"\t\t\t ---- Matricea originala ----"<<endl;

afisare(a,n,m);

int mx = INT\_MIN, indexI = -1, indexJ = -1;

for (int i = 1; i <= n; ++i) {

for (int j = 1; j <= m; ++j) {

if (a[i][j] > mx) {

mx = a[i][j];

indexI = i;

indexJ = j;

}

}

}

cout<<endl<<"Elementul maxim pentru care coloana si rindul vor fi eliminate [ "<<mx<<" ]";

if (indexI == n) {

--n;

} else {

for (int i = indexI; i < n; ++i)

for (int j = 1; j <= m; ++j)

a[i][j] = a[i + 1][j];

--n;

}

if (indexJ == m) {

--m;

} else {

for (int i = 1; i <= n; ++i)

for (int j = indexJ; j < m; ++j)

a[i][j] = a[i][j + 1];

--m;

}

cout<<endl<<"\t\t\tElementele matricii dupa stergere"<<endl;

afisare(a,n,m);

}

void Frecventa(int a[][101],int n,int m){

MN=abs(MN);

int freq[MX+1]={0};

int freqN[MN+1]={0};

for(int i=1;i<=n;i++)

for(int j=1;j<=m;j++)

if(a[i][j]<0)

freqN[abs(a[i][j])]++;

else freq[a[i][j]]++;

int mx=freq[0];

int indexP=0;

int indexN=0;

for(int i=1;i<=MX;i++)

if(freq[i]>mx){

mx=freq[i];

indexP=i;}

for(int i=1;i<=MN;i++)

if(freqN[i]>mx){

mx=freqN[i];

indexN=i;}

if(indexN>indexP)

indexP=-indexN;

afisare(a,n,m);

cout <<"\nCel mai frecvent element al matricei este " <<indexP<< endl;

}

void MediaAritmetica(int a[][101],int n,int m){

double aux;

int cnt=0;

for(int i=1;i<=n;i++)

for(int j=1;j<=m;j++)

if(a[i][j]>=-10 && a[i][j]<=10)

aux+=a[i][j], cnt++;

aux/=cnt;

cout<<"\n\t\t ---------- Media elementelor cuprinse in intervalul [-10;10] = "<<aux<<" ----------\n";

}

void OrdoneazaRinduri(int a[][101],int n,int m){

cout<<"\t\t\t ---- Matricea originala ----"<<endl;

afisare(a,n,m);

for(int x=1;x<=n;x++)

for(int i=1;i<m;i++)

for(int j=i+1;j<=m;j++)

if(a[x][i]>a[x][j]) swap(a[x][i],a[x][j]);

cout<<"\t\t\t ---- Matricea sortata pe linii ----"<<endl;

afisare(a,n,m);

}

void ScrieSubsir(int a[][101],int n,int m){

cout<<"\t\t\t ---- Matricea originala ----"<<endl;

afisare(a,n,m);

int lung\_mx=0,inceput\_mx,sfirsit\_mx,lung=1,inceput,sfirsit;

for(int i=1;i<=n;i++){

lung\_mx=0,inceput=1,sfirsit=1;

for(int j=1;j<=m;j++)

if(a[i][j+1]>a[i][j])

lung++, sfirsit=j+1;

else{

if(lung>lung\_mx){

lung\_mx=lung;

inceput\_mx=inceput;

sfirsit\_mx=sfirsit;

}

lung=1;

inceput=j+1;

sfirsit=j+1;

}

cout<<"\t\t\t--- SUBSIRUL CRESCATOAR DE LUNGIME MAXIMA PE RANDUL "<<i<<"---"<<endl;

for(int x=inceput\_mx;x<=sfirsit\_mx;x++)

cout<<a[i][x]<<' ';

cout<<endl;

for(int x=inceput\_mx;x<=sfirsit\_mx;x++)

out<<a[i][x]<<' ';

out<<endl;

}

cout<<"\n \n-------Datele au fost transmise in fisier.-------\n";

}

void ParcurgeSpirala(int a[][101],int n,int m){

cout<<"------ Matricea ------\n"<<endl;

afisare(a,n,m);

int sir[100\*100]={0};

int index=0;

bool V[n][m]={false};

int dr[]={0,1,0,-1};

int dc[]={1,0,-1,0};

int X=1,Y=1, di=0;

for(int i=1;i<=n\*m;i++){

sir[++index]=a[X][Y];

V[X][Y]=true;

int x=X+dr[di];

int y=Y+dc[di];

if(1<=x && x<=n && 1<=y && y<=m && !V[x][y]){

X=x;

Y=y;

}

else{

di=(di+1)%4;

X+=dr[di];

Y+=dc[di];

}

}

cout<<"Parcurgerea matricei in spirala: ";

for(int i=1;i<=index;i++)

cout<<sir[i]<<' ';

cout<<endl;

bool adevar1=true;

int intrus1,intrus2;

bool adevar2=true;

int progresie=sir[2]-sir[1];

for(int i=3;i<index;i++){

if(sir[i+1]-progresie==sir[i] && adevar2)

continue;

else{

if(!adevar1){

adevar2=false,intrus2=sir[i];

break;}

else adevar1=false,intrus1=sir[i];

}

}

if(adevar1==true && adevar2==true)

cout<<"DA \n"<<sir[index];

else{

cout<<"NU\n";

cout<<intrus1<<' ';

if(adevar2==false) cout<<intrus2;

}

}

void SubmatriceMaxima(int a[][101],int n,int m){

cout<<"------ Matricea ------\n"<<endl;

afisare(a,n,m);

cout<<endl;

cout<<"------ Submatricea de lungime maxima ------\n"<<endl;

cout<<"Coordonatele varfului din stinga sus: "<<"[0]"<<" [0]"<<endl;

if(n>m)

n-=(n-m);

else if(m>n)

m-=(m-n);

else{

--n;

--m;

}

afisare(a,n,m);

}

int main(){

CitesteFisier();

cout << "----------------------------------------" << endl;

cout << "| BUN VENIT! |" << endl;

cout << "----------------------------------------" << endl;

cout << "----------------------------------------" << endl;

afisare(A, n, m);

cout << "----------------------------------------" << endl;

bool continua=true;

do{

int optiune;

cout<<"\n1. Insereaza in matricea A o noua linie/coloana"<<endl;

cout<<"2. Exclude din matricea A linia si coloana la intersectia caror se afla primul element maximal intilnit la parcurgerea pe linii"<<endl;

cout<<"3. Gaseste numarul care apare cel mai frecvent in matricea A"<<endl;

cout<<"4. Determina media aritmetica a elementelor matricei A, care apartin intervalului [-10,10]"<<endl;

cout<<"5. Sorteaza crescator elementele fiecarei linii ale matricei A prin metoda numararii"<<endl;

cout<<"6. Copie subsirurile crescatoare de lungime maximala din liniile matricei A"<<endl;

cout<<"7. Gaseste in componenta matricei A submatricea patratica simetrica de dimensiune maximala"<<endl;

cout<<"8. Parcurge in spirala matricea si determina daca formeaza o progresie arimetica"<<endl;

cout<<"Selecteaza o optiune: ";

cin>>optiune;

switch(optiune){

case 1:

adauga(A,n,m);

break;

case 2:

StergeMaxim(A,n,m);

break;

case 3:

Frecventa(A,n,m);

break;

case 4:

MediaAritmetica(A,n,m);

break;

case 5:

OrdoneazaRinduri(A,n,m);

break;

case 6:

ScrieSubsir(A,n,m);

break;

case 7:

SubmatriceMaxima(A,n,m);

break;

case 8:

for(int i=1;i<=n;i++){

for(int j=1;j<=m;j++)

cout<<A[i][j]<<' ';

cout<<endl;

}

cout<<endl;

ParcurgeSpirala(A,n,m);

break;

case 0: continua=false; break;

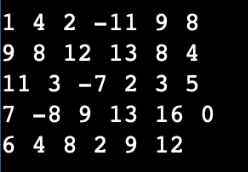
}

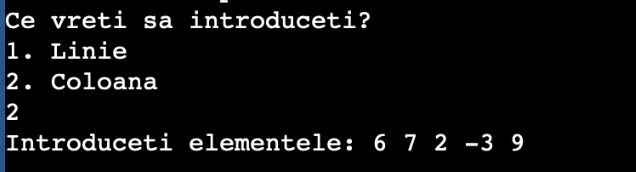
}while(continua);

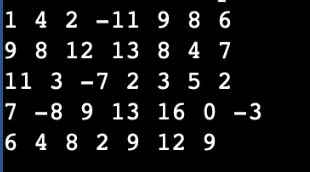
}

## Rezultatele testării

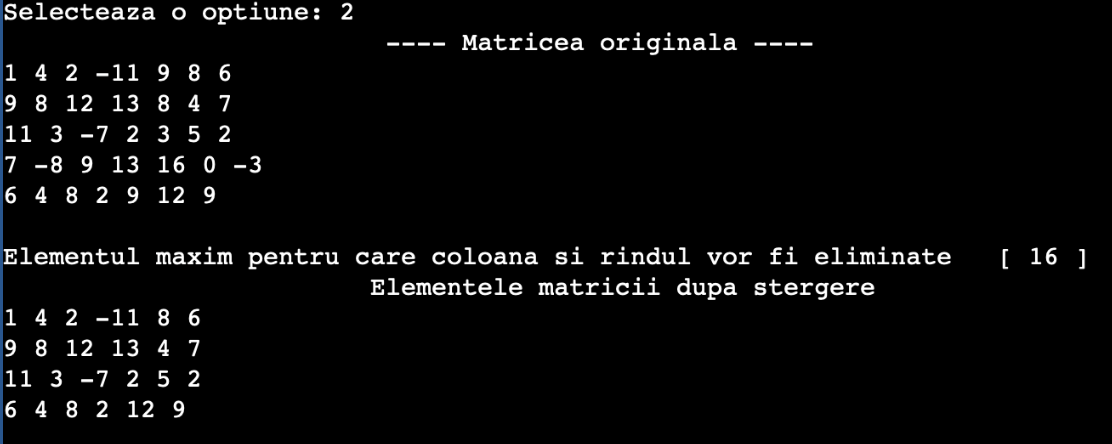
INTRODUCEREA UNEI LINII SAU COLOANE NOI



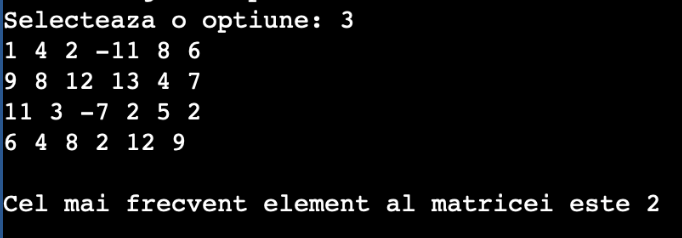




Exclude din matricea A linia si coloana la intersectia caror se afla primul element maximal intilnit la parcurgerea pe linii



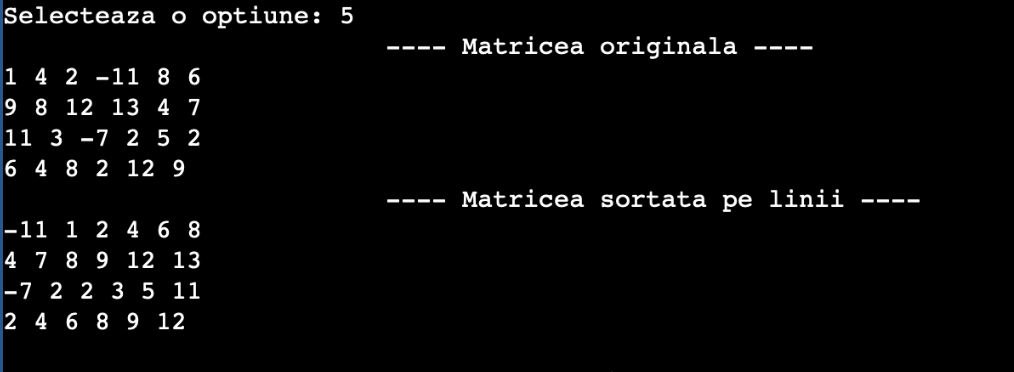
3. Gaseste numarul care apare cel mai frecvent in matricea A



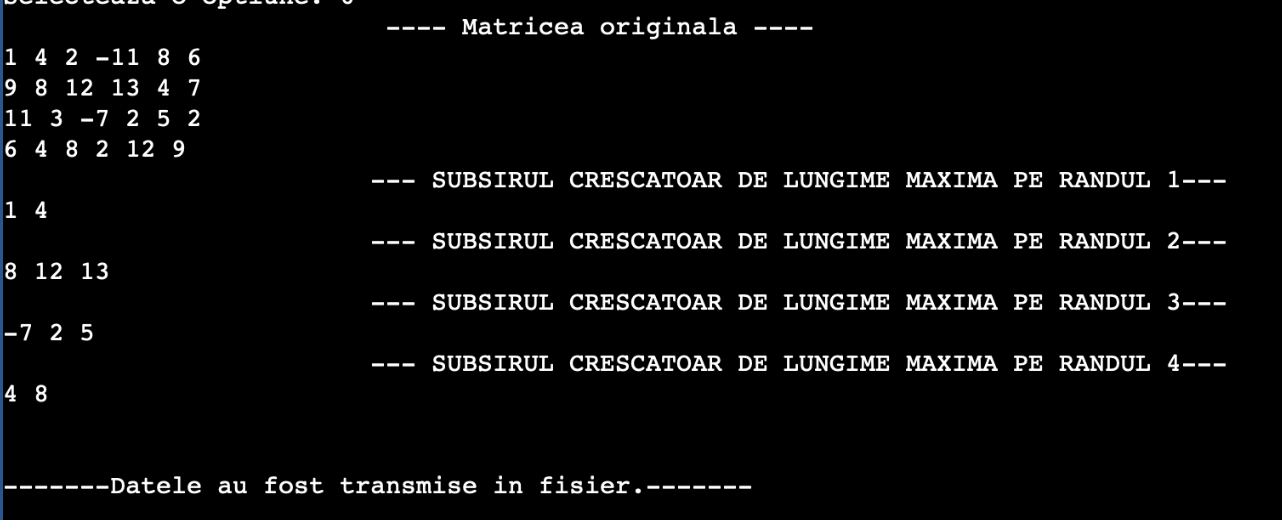
4. Determina media aritmetica a elementelor matricei A, care apartin intervalului [-10,10]



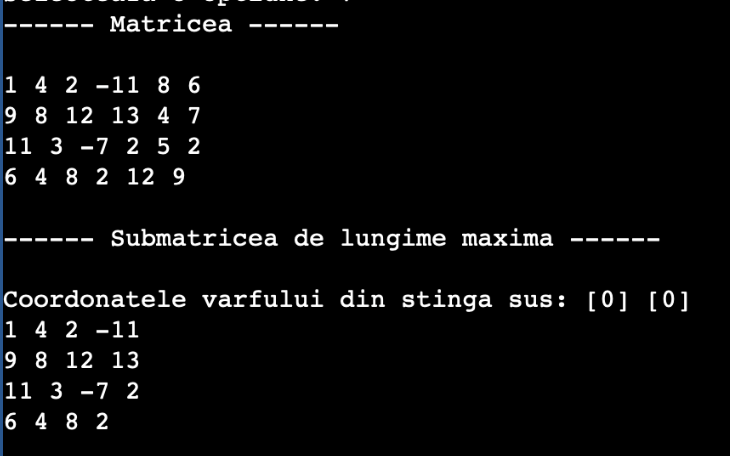
5. Sorteaza crescator elementele fiecarei linii ale matricei A prin metoda numararii



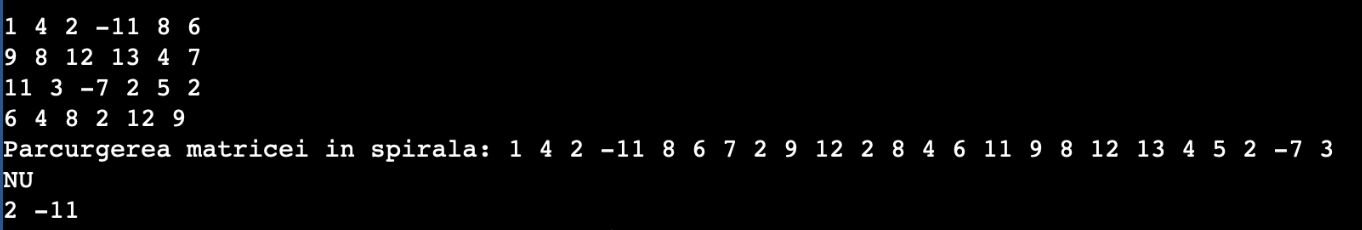
6. Copie subsirurile crescatoare de lungime maximala din liniile matricei A



7. Gaseste in componenta matricei A submatricea patratica simetrica de dimensiune maximala



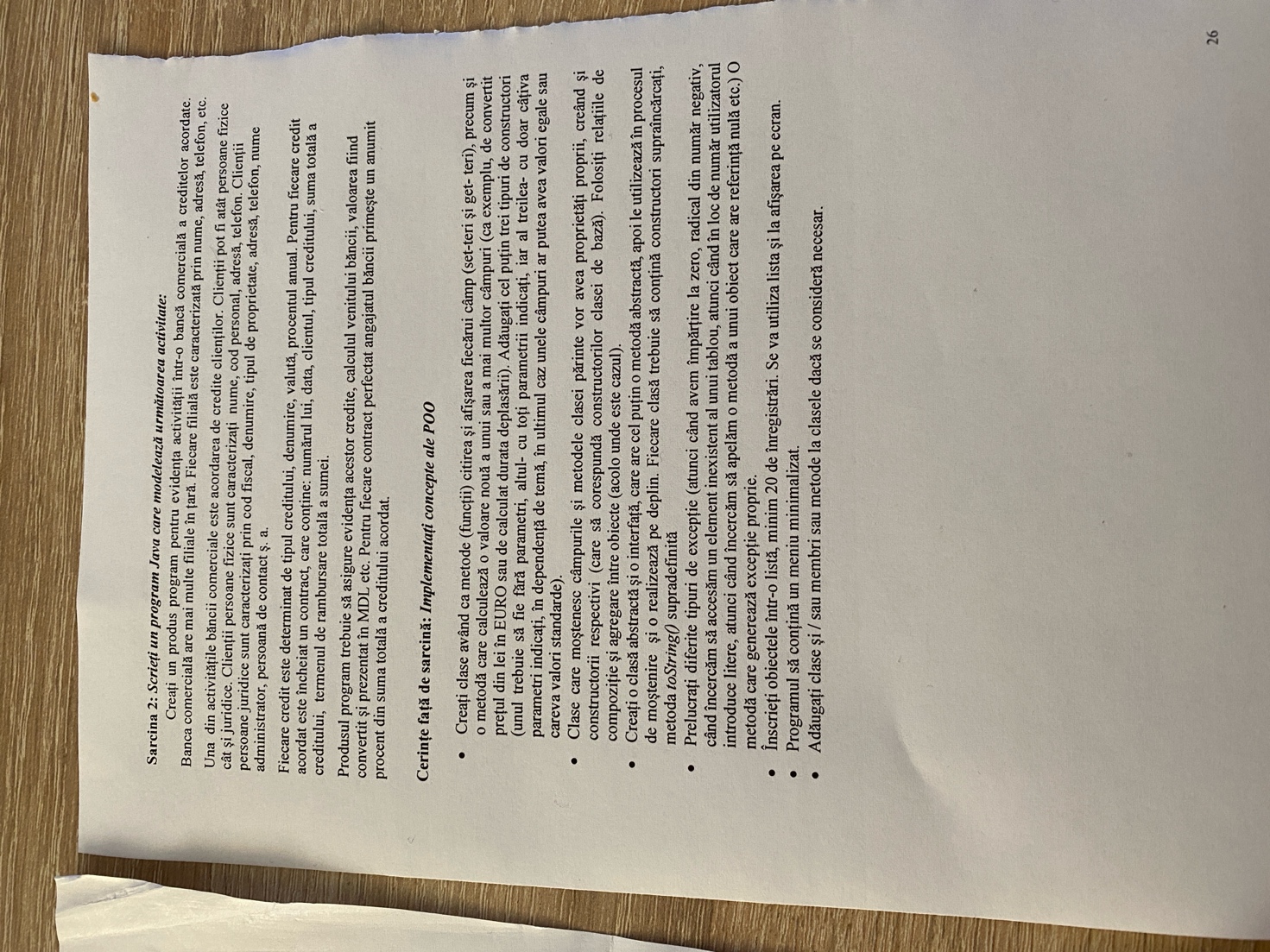
8. Parcurge in spirala matricea si determina daca formeaza o progresie arimetica



### Date de intrare



# Conținutul activităților și sarcinilor de lucru (Java)



## Descrierea modului de elaborare a aplicației

În procesul dezvoltării aplicației bancare, am parcurs mai multe etape esențiale pentru a asigura o implementare eficientă și funcțională. Am început prin definirea și structurarea claselor principale, inclusiv a interfețelor necesare pentru gestionarea creditelor. Am aplicat principiile de încapsulare prin definirea variabilelor private și prin crearea metodelor getter și setter pentru accesul controlat la date. De asemenea, am suprascris metoda toString() pentru a asigura o reprezentare clară a obiectelor în text.

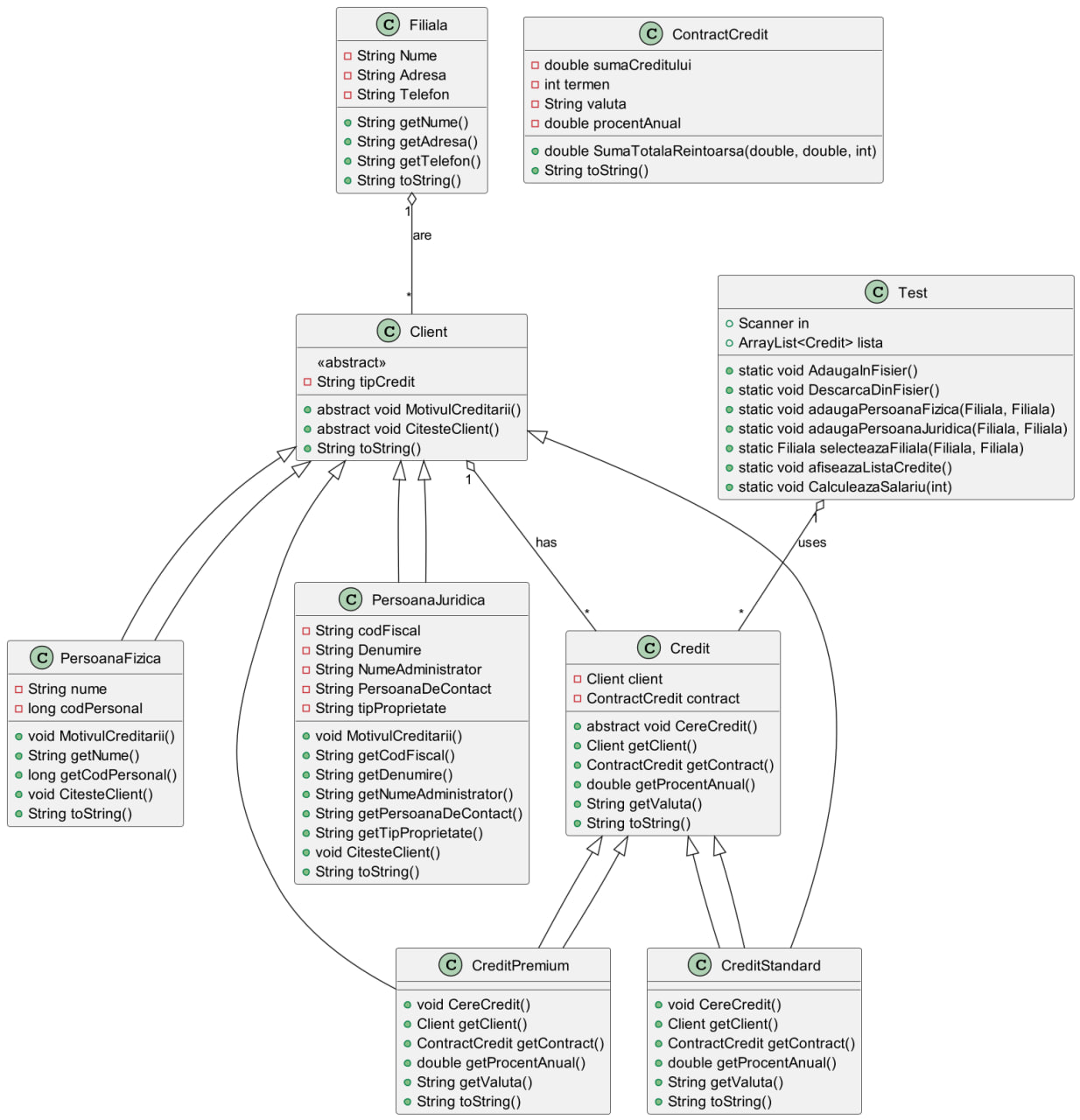
În continuare, am implementat funcționalitățile specifice cerințelor proiectului. Am utilizat clasele abstracte și interfețele pentru a permite o gestionare flexibilă și eficientă a creditelor. Clasele concrete CreditStandard și CreditPremium au fost dezvoltate pentru a extinde comportamentul clasei abstracte Credit și pentru a adăuga logica specifică pentru fiecare tip de credit.

Am integrat aceste clase și interfețe în aplicația principală (Test), unde am creat o interfață de utilizator simplă pentru a permite adăugarea de clienți și gestionarea creditelor. Am implementat un sistem de meniu interactiv care oferă opțiuni pentru adăugarea de credite standard și premium, afișarea listei de credite și calculul veniturilor asociate operațiunilor bancare.

În timpul implementării, am acordat o atenție deosebită gestionării excepțiilor și validării datelor introduse de utilizator, asigurându-ne că aplicația funcționează corect în diferite scenarii de utilizare. Am testat și am validat fiecare funcționalitate implementată pentru a asigura funcționarea corespunzătoare și pentru a îmbunătăți experiența utilizatorului.

În concluzie, prin aplicarea principiilor de proiectare orientată pe obiecte, prin utilizarea claselor abstracte și interfețelor, precum și prin atenția acordată detaliilor de implementare și testare, am reușit să dezvoltăm o aplicație bancară robustă și eficientă pentru gestionarea creditelor, adaptată nevoilor și cerințelor proiectului nostru.

### Proiectarea UML a sistemului de management



### Descrierea claselor și interfețelor

|  |  |
| --- | --- |
| Clasa Client servește ca o șablonă pentru reprezentarea datelor unui client în sistemul vostru de gestionare a creditelor. Ea stochează informații esențiale cum ar fi adresa, numărul de telefon și detaliile despre creditul solicitat. Prin intermediul metodelor getter și setter, clasele concrete pot accesa și actualiza aceste informații într-un mod controlat.  Implementând interfața Serializable, clasa permite obiectelor sale să fie serializate și deserializate, facilitând astfel stocarea și transferul de date într-un format ușor de gestionat.  Metoda toString() este suprascrisă pentru a oferi o reprezentare coerentă și estetică a informațiilor despre client atunci când sunt afișate, folosind și formatarea colorată pentru o prezentare vizuală plăcută.  În esență, Client reprezintă o structură fundamentală în arhitectura aplicației voastre, oferind un cadru robust pentru gestionarea datelor de client într-un mod eficient și organizat. | import java.io.Serializable; import java.time.LocalDate; import java.util.List; import java.util.Scanner;  interface FormularCreditare{  void MotivulCreditarii();  }  abstract class Client implements Serializable, FormularCreditare {  transient Scanner in=new Scanner(System.*in*);  protected String Adresa,telefon, tipCredit;   protected Filiala filiala;     public String getTipCredit() {  return tipCredit;  }   public void setTipCredit(String tipCredit) {  this.tipCredit = tipCredit;  }   abstract void CitesteClient();    public Filiala getFiliala() {  return filiala;  }   public void setFiliala(Filiala filiala) {  this.filiala = filiala;  }    public String getAdresa() {  return Adresa;  }   public void setAdresa(String adresa) {  Adresa = adresa;  }   public String getTelefon() {  return telefon;  }   public void setTelefon(String telefon) {  this.telefon = telefon;  }   @Override  public String toString() {  return String.*format*(  "\033[1;34mClient:\033[0m\n" +  "\033[1;33m Adresa:\033[0m %s\n" +  "\033[1;33m Telefon:\033[0m %s\n" +  "\033[1;33m Filiala:\033[0m %s\n" +  "\033[1;33m Tip Credit:\033[0m %s\n",  Adresa, telefon, filiala, tipCredit  );  }    } |
| Clasa Credit servește ca o entitate abstractă pentru gestionarea informațiilor legate de un credit. Aceasta conține date esențiale precum filiala asociată creditului, detalii despre contractul de credit, informații despre clientul beneficiar al creditului și detalii specifice despre valută și dobânzi. Metodele abstracte din clasa sunt definite pentru calcularea ratei lunare, conversia sumelor în diverse valute și calcularea sumei totale de rambursat, facilitând astfel manipularea și calculul detaliilor financiare specifice fiecărui tip de credit implementat. | import java.io.Serializable; import java.time.LocalDate; import java.util.List; import java.util.Scanner;  abstract public class Credit implements Serializable {  transient Scanner in=new Scanner(System.*in*);   Filiala filiala;  ContractCredit contract;  Client client;  private String Valuta;  private double procentAnual;  private double procentLunar;   public Filiala getFiliala() {  return filiala;  }   public void setFiliala(Filiala filiala) {  this.filiala = filiala;  }   public double getProcentLunar() {  return procentLunar;  }   public void setProcentLunar(double procentLunar) {  this.procentLunar=procentLunar;  }   abstract void CereCredit();   public Client getClient() {  return client;  }   public void setClient(Client client) {  this.client = client;  }     abstract double calculRataLunara(double suma, int termen,double procentAnual);  abstract double ConvertireValuta(String valuta, double suma);  abstract double SumaTotalaReintoarsa(double SumaInitiala,double Procent,int termen);    public ContractCredit getContract() {  return contract;  }   public void setContract(ContractCredit contract) {  this.contract = contract;  }     public String getValuta() {  return Valuta;  }   public void setValuta(String valuta) {  Valuta = valuta;  }   public double getProcentAnual() {  return procentAnual;  }   public void setProcentAnual(double procentAnual) {  this.procentAnual = procentAnual;  }   @Override  public String toString() {  StringBuilder sb = new StringBuilder();  sb.append("\n==============================\n")  .append(" DETALII CREDIT \n")  .append("==============================\n")  .append("Client: ").append(client != null ? client.toString() : "N/A").append('\n')  .append("Valuta: ").append(getValuta()).append('\n')  .append("Procent Anual: ").append(getProcentAnual()).append('%').append('\n')  .append("Procent Lunar: ").append(getProcentLunar()).append('%').append('\n')  .append("Contract: ").append(contract != null ? contract.toString() : "N/A").append('\n')  .append("Rata Lunară: ").append(String.*format*("%.2f %s", calculRataLunara(contract.getSumaCreditului(), contract.getTermen(), getProcentAnual()), getValuta())).append('\n')  .append("Suma Totală de Reîntoarcere: ").append(String.*format*("%.2f %s", SumaTotalaReintoarsa(contract.getSumaCreditului(), getProcentAnual(), contract.getTermen()), getValuta())).append('\n')  .append("==============================\n");  return sb.toString();  }   Credit(Client client){  this.client=client;  } } |
| Clasa CreditStandard reprezintă o implementare specifică a clasei abstracte Credit, destinată gestionării unui tip standard de credit. Aceasta definește comportamente și caracteristici specifice pentru procesul de cerere și gestionare a creditului standard.  În constructorul său, CreditStandard primește un client și inițializează procentul anual la 20%. Metoda CereCredit()permite utilizatorului să introducă detalii despre cererea de credit, inclusiv valuta (Lei sau Euro), suma dorită și termenul în luni. Aceasta validează intrările utilizatorului, aplică restricțiile specifice pentru persoane fizice în funcție de valută și afișează un mesaj de confirmare la finalizarea cererii.  Metoda calculRataLunara() calculează rata lunară pentru credit pe baza sumei, termenului și procentului anual. De asemenea, clasa oferă o implementare simplă a metodei ConvertireValuta() pentru convertirea sumelor din Euro în Lei.  Metoda SumaTotalaReintoarsa() calculează suma totală de rambursat pentru credit, bazată pe rata lunară calculată și termenul în luni.  Această implementare a clasei CreditStandard servește drept schelet flexibil pentru gestionarea detaliilor specifice unui credit standard, inclusiv calculul ratelor și conversiile valutare necesare în contextul financiar specificat. | import java.time.LocalDate; import java.util.\*;  public class CreditStandard extends Credit {   {  this.setProcentAnual(20);  }   CreditStandard(Client client) {  super(client);  }    @Override  void CereCredit() {  Scanner in = new Scanner(System.*in*);   System.*out*.println("\n=====================================");  System.*out*.println(" CERERE CREDIT - DETALII CONTRACT ");  System.*out*.println("=====================================");   // Alegere valuta  String valuta = "";  boolean validValuta = false;  while (!validValuta) {  System.*out*.print("Alegeți valuta (Lei/Euro): ");  valuta = in.next();  if (valuta.equalsIgnoreCase("Lei") || valuta.equalsIgnoreCase("Euro")) {  validValuta = true;  } else {  System.*out*.println("\u001B[31mValuta invalidă. Te rugăm să alegi între Lei și Euro.\u001B[37m");  }  }  setValuta(valuta);    System.*out*.print("Suma creditului (" + valuta + "): ");  double sumaCredit = 0;  boolean validSuma = false;  while (!validSuma) {  try {  sumaCredit = in.nextDouble();   if (getClient() instanceof PersoanaFizica) {  if (valuta.equalsIgnoreCase("Lei") && sumaCredit > 50000) {  throw new IllegalArgumentException("Suma maximă pentru persoane fizice este de 50,000 lei.");  } else if (valuta.equalsIgnoreCase("Euro") && sumaCredit > 3000) {  throw new IllegalArgumentException("Suma maximă pentru persoane fizice este de 3,000 euro.");  }  }   if (sumaCredit <= 0) {  throw new IllegalArgumentException("Suma trebuie să fie mai mare decât zero.");  }  validSuma = true;  } catch (InputMismatchException e) {  System.*out*.println("\u001B[31mEroare la introducerea sumei.\u001B[37m");  System.*out*.print("Introduceți din nou suma: ");  in.nextLine();  } catch (IllegalArgumentException e) {  System.*out*.println("\u001B[31m" + e.getMessage() + "\u001B[37m");  System.*out*.print("Introduceți din nou suma: ");  in.nextLine();  }  }    System.*out*.print("Termenul creditului (în luni): ");  int termenCredit = 0;  boolean validTermen = false;  while (!validTermen) {  try {  termenCredit = in.nextInt();  if (termenCredit <= 0) {  throw new IllegalArgumentException("Termenul trebuie să fie mai mare decât zero.");  }   validTermen = true;  } catch (InputMismatchException e) {  System.*out*.println("\u001B[31mEroare la introducerea termenului.\u001B[37m");  System.*out*.print("Introduceți din nou termenul (în luni): ");  in.nextLine();  } catch (IllegalArgumentException e) {  System.*out*.println("\u001B[31m" + e.getMessage() + "\u001B[37m");  System.*out*.print("Introduceți din nou termenul (în luni): ");  in.nextLine();  }  }   ContractCredit contract = new ContractCredit();  Random random = new Random();  long nrContract = 1000000000L + (long) (random.nextDouble() \* 9000000000L);  contract.setNrCredit(nrContract);  contract.setSumaCreditului(sumaCredit);  contract.setTermen(termenCredit);  contract.setDataIntemeierii(LocalDate.*now*());  List<LocalDate> planRambursare = new ArrayList<>();  contract.setPlanRambursare(planRambursare, termenCredit);   setContract(contract);    setProcentLunar(getProcentAnual()/termenCredit);   System.*out*.println("\n=============================");  System.*out*.println(" Cerere credit finalizată!");  System.*out*.println("=============================\n");  }   @Override  double calculRataLunara(double suma, int termen, double procentAnual) {  double procentLunar = procentAnual / 100 / 12;   double rataLunaraneto=suma/termen;   rataLunaraneto+=rataLunaraneto\*procentLunar;   return rataLunaraneto;   }   double ConvertireValuta(String valuta, double suma) {  double rezultat = suma;   switch (valuta) {  case "Euro":  rezultat = suma \* 19.12;  break;  default:  System.*out*.println("Valuta introdusă nu este suportată pentru conversie în Lei.");  break;  }   return rezultat;  }   @Override  double SumaTotalaReintoarsa(double SumaInitiala,double procent, int termen) {   double sumaLunara= calculRataLunara(SumaInitiala, termen, procent);  return sumaLunara\*termen+SumaInitiala;  }    } |
| Clasa CreditPremium este o extensie a clasei abstracte Credit, specializată în gestionarea creditelor premium. Aceasta include funcționalități pentru cererea de credit, calculul ratelor lunare și conversia sumelor în lei din mai multe valute (Euro, Dolari, Roni). Implementarea sa se bazează pe o inițializare a procentului anual la 12%, fiind adaptată pentru oferirea de servicii premium într-un context financiar. | import java.time.LocalDate; import java.util.ArrayList; import java.util.InputMismatchException; import java.util.List; import java.util.Random;  public class CreditPremium extends Credit{   {  this.setProcentAnual(12);  }   CreditPremium(Client client) {  super(client);  }   @Override  void CereCredit() {   System.*out*.println("\n=====================================");  System.*out*.println(" CERERE CREDIT - DETALII CONTRACT ");  System.*out*.println("=====================================");   // Alegere valuta  String valuta = "";  boolean validValuta = false;  while (!validValuta) {  System.*out*.print("Alegeți valuta (Lei/Euro/Dolari/Roni): ");  valuta = in.next();  if (valuta.equalsIgnoreCase("Lei") || valuta.equalsIgnoreCase("Euro") || valuta.equalsIgnoreCase("Dolari") || valuta.equalsIgnoreCase("Roni")) {  validValuta = true;  } else {  System.*out*.println("\u001B[31mValuta invalidă. Te rugăm să alegi între Lei, Euro, Dolari și Roni.\u001B[37m");  }  }  setValuta(valuta);   // Suma creditului  System.*out*.print("Suma creditului (" + valuta + "): ");  double sumaCredit = 0;  boolean validSuma = false;  while (!validSuma) {  try {  sumaCredit = in.nextDouble();  if (sumaCredit <= 0) {  throw new IllegalArgumentException("Suma trebuie să fie mai mare decât zero.");  }  validSuma = true;  } catch (InputMismatchException e) {  System.*out*.println("\u001B[31mEroare la introducerea sumei.\u001B[37m");  System.*out*.print("Introduceți din nou suma: ");  in.nextLine(); // Consumăm newline-ul rămas în buffer  } catch (IllegalArgumentException e) {  System.*out*.println("\u001B[31m" + e.getMessage() + "\u001B[37m");  System.*out*.print("Introduceți din nou suma: ");  in.nextLine(); // Consumăm newline-ul rămas în buffer  }  }   // Termenul creditului  System.*out*.print("Termenul creditului (în luni): ");  int termenCredit = 0;  boolean validTermen = false;  while (!validTermen) {  try {  termenCredit = in.nextInt();  if (termenCredit <= 0) {  throw new IllegalArgumentException("Termenul trebuie să fie mai mare decât zero.");  }  validTermen = true;  } catch (InputMismatchException e) {  System.*out*.println("\u001B[31mEroare la introducerea termenului.\u001B[37m");  System.*out*.print("Introduceți din nou termenul (în luni): ");  in.nextLine(); // Consumăm newline-ul rămas în buffer  } catch (IllegalArgumentException e) {  System.*out*.println("\u001B[31m" + e.getMessage() + "\u001B[37m");  System.*out*.print("Introduceți din nou termenul (în luni): ");  in.nextLine(); // Consumăm newline-ul rămas în buffer  }  }   // Citirea detaliilor suplimentare din contract  ContractCredit contract = new ContractCredit();  Random random = new Random();  long nrContract = 1000000000L + (long) (random.nextDouble() \* 9000000000L);  contract.setNrCredit(nrContract);  contract.setSumaCreditului(sumaCredit);  contract.setTermen(termenCredit);  contract.setDataIntemeierii(LocalDate.*now*());  List<LocalDate> planRambursare = new ArrayList<>();  contract.setPlanRambursare(planRambursare, termenCredit);   setContract(contract);   setProcentLunar(getProcentAnual()/termenCredit);   System.*out*.println("\n=============================");  System.*out*.println(" Cerere credit finalizată!");  System.*out*.println("=============================\n");  }   @Override  double calculRataLunara(double suma, int termen, double procentAnual) {  double procentLunar = procentAnual / 100 / 12;   double rataLunaraneto=suma/termen;   rataLunaraneto+=rataLunaraneto\*procentLunar;   return rataLunaraneto;   }    double ConvertireValuta(String valuta, double suma) {  double rezultat = suma;   switch (valuta) {  case "Euro":  rezultat = suma \* 19.12;  break;  case "Dolar":  rezultat = suma \* 17.90;  break;  case "Ron":  rezultat = suma \* 3.93;  break;   default:  System.*out*.println("Valuta introdusă nu este suportată pentru conversie în Lei.");  break;  }   return rezultat;  }   @Override  double SumaTotalaReintoarsa(double SumaInitiala, double Procent, int termen) {   double sumaLunara= calculRataLunara(SumaInitiala, termen, Procent\*1.2);  return sumaLunara\*termen+SumaInitiala;  }     } |
| Clasa ContractCredit este responsabilă pentru gestionarea și stocarea detaliilor legate de un contract de credit în cadrul unei aplicații Java. Ea include atribute pentru numărul unic al contractului (nrCredit), suma creditului (SumaCreditului), termenul în luni (termen), data de încheiere a contractului (DataIntemeierii), și o listă de obiecte LocalDate care reprezintă planul de rambursare al ratelor lunare (PlanRambursare).  Constructorul său inițializează lista PlanRambursare, iar metoda setPlanRambursare este folosită pentru a calcula și seta detaliile planului de rambursare pe baza termenului contractului, începând de la o lună după data de încheiere.  Clasa include și o metodă citesteContract care permite utilizatorului să introducă detaliile contractului de la tastatură, inclusiv suma creditului, termenul și data de încheiere. Metoda toString este suprascrisă pentru a oferi o reprezentare formatată a detaliilor contractului, inclusiv numărul de contract, suma creditului, data de încheiere și fiecare rată lunară din planul de rambursare, toate afișate într-un format clar și ușor de înțeles. Această clasă este utilă în contexte financiare și permite manipularea eficientă a detaliilor contractuale în aplicațiile Java. | import java.io.Serializable; import java.time.\*; import java.time.format.DateTimeFormatter; import java.util.ArrayList; import java.util.List; import java.util.Scanner;  public class ContractCredit implements Serializable {  private long nrCredit;  private double SumaCreditului;  private int termen;  private LocalDate DataIntemeierii;  private List<LocalDate> PlanRambursare;   public ContractCredit() {  PlanRambursare = new ArrayList<>();  }   public int getTermen() {  return termen;  }   public void setTermen(int termen) {  this.termen = termen;  }   public long getNrCredit() {  return nrCredit;  }   public void setNrCredit(long nrCredit) {  this.nrCredit = nrCredit;  }   public double getSumaCreditului() {  return SumaCreditului;  }   public void setSumaCreditului(double sumaCreditului) {  SumaCreditului = sumaCreditului;  }   public LocalDate getDataIntemeierii() {  return DataIntemeierii;  }   public void setDataIntemeierii(LocalDate dataIntemeierii) {  DataIntemeierii = dataIntemeierii;  }   public List<LocalDate> getPlanRambursare() {  return PlanRambursare;  }   public void setPlanRambursare(List<LocalDate> planRambursare, int termen) {  PlanRambursare.clear();   LocalDate dataCurenta = DataIntemeierii.plusMonths(1); // Prima rata dupa o luna de la data intemeierii   for (int i = 0; i < termen; i++) {  PlanRambursare.add(dataCurenta);  dataCurenta = dataCurenta.plusMonths(1);  }  }   public void citesteContract(Scanner in) {  System.*out*.println("\n==============================");  System.*out*.println(" CITIRE DATE CONTRACT ");  System.*out*.println("==============================");   System.*out*.print("Introduceți suma creditului: ");  double sumaCreditului = in.nextDouble();   System.*out*.print("Introduceți termenul (în luni): ");  int termen = in.nextInt();  in.nextLine();   System.*out*.print("Introduceți data întemeierii (YYYY-MM-DD): ");  String data = in.nextLine();  LocalDate dataIntemeierii = LocalDate.*parse*(data);   setSumaCreditului(sumaCreditului);  setTermen(termen);  setDataIntemeierii(dataIntemeierii);   setPlanRambursare(PlanRambursare, termen);  }   @Override  public String toString() {  DateTimeFormatter formatter = DateTimeFormatter.*ofPattern*("dd MMMM yyyy");   StringBuilder sb = new StringBuilder();  sb.append("\n==============================\n")  .append(" DETALII CONTRACT \n")  .append("==============================\n")  .append("Număr Credit: ").append(nrCredit).append('\n')  .append("Suma Creditului: ").append(SumaCreditului).append('\n')  .append("Data Întemeierii: ").append(DataIntemeierii.format(formatter)).append('\n')  .append("Plan de Rambursare:\n");   for (int i = 0; i < PlanRambursare.size(); i++) {  sb.append("Rata ").append(i + 1).append(": ").append(PlanRambursare.get(i).format(formatter)).append('\n');  }   sb.append("==============================\n");   return sb.toString();  } } |
| Clasa Filiala definește o entitate care reprezintă o filială într-un sistem Java, cu atributele ei principale: Nume, Adresa, și Telefon. Această clasă implementează interfața Serializable, ceea ce permite obiectelor să fie serializate pentru stocare și transmitere ușoară.  Constructorii clasei permit inițializarea obiectului Filiala cu informațiile necesare sau pot fi folosiți pentru a crea o instanță goală. Metodele get și set sunt utilizate pentru a accesa și modifica aceste atribute.  Metoda toString este suprascrisă pentru a oferi o reprezentare detaliată și estetică a obiectului Filiala. Ea încadrează informațiile despre filială într-un cadru format din caractere \* albastre, cu textul centralizat și formatat pentru a fi ușor de citit și de înțeles.  Această clasă este utilă în scenariile în care este necesară manipularea și afișarea detaliilor despre filiale într-un mod organizat și estetic în aplicațiile Java. | import java.io.Serializable;  class Filiala implements Serializable {  private String Nume, Adresa, Telefon;   public String getNume() {  return Nume;  }   public void setNume(String nume) {  Nume = nume;  }   public String getAdresa() {  return Adresa;  }   public void setAdresa(String adresa) {  Adresa = adresa;  }   public String getTelefon() {  return Telefon;  }   public void setTelefon(String telefon) {  Telefon = telefon;  }   public Filiala(String nume, String adresa, String telefon) {  Nume = nume;  Adresa = adresa;  Telefon = telefon;  }   Filiala() {  }   private static final String *ANSI\_RESET* = "\u001B[0m";  private static final String *ANSI\_BLUE* = "\u001B[34m";   @Override  public String toString() {  int frameWidth = 50;  StringBuilder sb = new StringBuilder();  sb.append("Filiala").append('\n');  sb.append("Nume: ").append(Nume).append('\n');  sb.append("Adresa: ").append(Adresa).append('\n');  sb.append("Telefon: ").append(Telefon).append('\n');   String[] lines = sb.toString().split("\n");  StringBuilder centered = new StringBuilder();  for (String line : lines) {  int padding = (frameWidth - line.length()) / 2;  centered.append(" ".repeat(padding)).append(line).append('\n');  }   String border = *ANSI\_BLUE* + "\*".repeat(frameWidth) + *ANSI\_RESET*;  StringBuilder framedText = new StringBuilder(border).append('\n');  for (String line : centered.toString().split("\n")) {  String framedLine = *ANSI\_BLUE* + "\*" + *ANSI\_RESET* + line + " ".repeat(frameWidth - line.length() - 2) + *ANSI\_BLUE* + "\*" + *ANSI\_RESET*;  framedText.append(framedLine).append('\n');  }  framedText.append(border);   return framedText.toString();  }    } |
| Clasa PersoanaFizica reprezintă o entitate specifică de client în cadrul sistemului tău, extinzând clasa abstractă Client. Aceasta adaugă atributele nume și codPersonal pentru a caracteriza o persoană fizică în contextul solicitării unui credit.  Metoda MotivulCreditarii este suprascrisă din interfața FormularCreditare, oferind utilizatorului opțiuni pentru a selecta motivul pentru care dorește creditarea. Aceasta utilizează un obiect Scanner (in) pentru a permite introducerea opțiunilor și a detaliilor suplimentare în cazul opțiunii "Altele".  Metoda CitesteClient este de asemenea suprascrisă pentru a solicita și a stoca informațiile specifice unei persoane fizice, cum ar fi numele, codul personal, adresa și telefonul. Aceasta utilizează și metoda MotivulCreditarii pentru a completa detaliile necesare pentru solicitarea de credit.  Metoda toString oferă o reprezentare formatată în mod estetic a obiectului PersoanaFizica, incluzând numele, codul personal și detalii suplimentare despre client și filiala asociată.  Această clasă este utilă în contextul în care aplicația ta trebuie să gestioneze informațiile specifice ale unei persoane fizice în legătură cu procesul de creditare. Ea extinde funcționalitatea clasei de bază Client și implementează comportamente specifice unei persoane fizice în cadrul sistemului tău. | import java.util.ArrayList;  public class PersoanaFizica extends Client{  private String nume;  private long codPersonal;     @Override  public void MotivulCreditarii() {  System.*out*.println("\033[1;36mCare este motivul creditarii?\033[0m");  System.*out*.println("1. Vacanta.");  System.*out*.println("2. Bunuri materiale.");  System.*out*.println("3. Achizitioarea unui automobil / locuinte.");  System.*out*.println("4. Altele.");   int optiune = in.nextInt();  in.nextLine();   switch (optiune) {  case 1:  setTipCredit("Vacanta");  break;  case 2:  setTipCredit("Bunuri materiale");  break;  case 3:  setTipCredit("Achizitioarea unui automobil / locuinte");  break;  case 4:  System.*out*.print("Introduceți motivul creditării: ");  setTipCredit(in.nextLine());  break;  default:  System.*out*.println("Opțiune invalidă. Vă rugăm să alegeți o opțiune validă.");  MotivulCreditarii();  break;  }  }     public String getNume() {  return nume;  }   public void setNume(String nume) {  this.nume = nume;  }   public long getCodPersonal() {  return codPersonal;  }   public void setCodPersonal(long codPersonal) {  this.codPersonal = codPersonal;  }   @Override  void CitesteClient() {  System.*out*.print("\033[1;32mIntroduceți numele:\033[0m ");  setNume(in.nextLine());   System.*out*.print("\033[1;32mIntroduceți codul personal:\033[0m ");  setCodPersonal(in.nextLong());  in.nextLine();   System.*out*.print("\033[1;32mIntroduceți adresa:\033[0m ");  setAdresa(in.nextLine());   System.*out*.print("\033[1;32mIntroduceți telefonul:\033[0m ");  setTelefon(in.nextLine());   System.*out*.println("\033[1;32mIntroduceți detaliile filialei:\033[0m");  filiala = new Filiala();   MotivulCreditarii();  }    @Override  public String toString() {  return String.*format*(  "\033[1;31mPersoana Fizica:\033[0m\n" +  "\033[1;33m Nume:\033[0m %s\n" +  "\033[1;33m Cod Personal:\033[0m %d\n%s",  nume, codPersonal, super.toString()  );  } } |
| Clasa PersoanaJuridica reprezintă o entitate specifică de client în cadrul sistemului tău de creditare, extinzând clasa abstractă Client. Aceasta include atributele specifice unei persoane juridice necesare pentru gestionarea procesului de creditare.  Atributele clasei includ:   * codFiscal: Codul fiscal al persoanei juridice. * Denumire: Denumirea sau numele oficial al persoanei juridice. * NumeAdministrator: Numele administratorului sau a persoanei responsabile în cadrul persoanei juridice. * PersoanaDeContact: Persoana de contact pentru persoana juridică. * tipProprietate: Tipul de proprietate asociat persoanei juridice, selectat dintr-o listă predefinită de opțiuni.   Metoda MotivulCreditarii este suprascrisă pentru a permite persoanelor juridice să specifice motivul pentru care doresc să solicite creditul. Utilizatorul poate alege din opțiuni prestabilite sau poate introduce un motiv personalizat.  Metoda CitesteClient gestionează introducerea și stocarea informațiilor necesare despre persoana juridică, inclusiv codul fiscal, denumirea, numele administratorului, persoana de contact, tipul de proprietate și detaliile de contact (adresă și telefon). Aceasta utilizează o metodă auxiliară afișațiMeniul pentru a afișa opțiunile disponibile pentru tipul de proprietate.  Metoda toString oferă o reprezentare formatată în mod estetic a obiectului PersoanaJuridica, incluzând toate informațiile relevante despre aceasta, alături de detaliile suplimentare ale filialei asociate.  Această clasă este utilă în implementarea unui sistem de creditare care să poată gestiona cererile și detaliile specifice ale persoanelor juridice, aducând în evidență nevoile și informațiile critice pentru procesul de evaluare și aprobare a creditelor. | import java.util.ArrayList; import java.util.InputMismatchException; import java.util.List;  public class PersoanaJuridica extends Client{   private String codFiscal, Denumire, NumeAdministrator, PersoanaDeContact;   public static List<String> *tipuriProprietati* = new ArrayList<>();   static{  *tipuriProprietati*.add("Proprietăți rezidențiale");  *tipuriProprietati*.add("Proprietăți comerciale");  *tipuriProprietati*.add("Proprietăți industriale");  *tipuriProprietati*.add("Proprietăți pentru uz special");  *tipuriProprietati*.add("Terenuri");   *tipuriProprietati*.add("Acțiuni");  *tipuriProprietati*.add("Obligațiuni");  *tipuriProprietati*.add("Instrumente de piață monetară");  *tipuriProprietati*.add("Fonduri de investiții");  *tipuriProprietati*.add("Derivate financiare");  *tipuriProprietati*.add("Proprietăți necorporale");  *tipuriProprietati*.add("Numerar și echivalente de numerar");  }  private String tipProprietate;   public String getTipProprietate() {  return tipProprietate;  }   public void setTipProprietate(String tipProprietate) {  this.tipProprietate = tipProprietate;  }   @Override  public void MotivulCreditarii() {  System.*out*.println("\033[1;36mCare este motivul pentru luarea acestui credit?\033[0m");  System.*out*.println("1. Investiție");  System.*out*.println("2. Grant");  System.*out*.println("3. Altele.");   int optiune;  try {  optiune = in.nextInt();  in.nextLine();   switch (optiune) {  case 1:  setTipCredit("Investiție");  break;  case 2:  setTipCredit("Grant");  break;  case 3:  System.*out*.print("Introduceți motivul creditării: ");  setTipCredit(in.nextLine());  break;  default:  System.*out*.println("Opțiune invalidă. Vă rugăm să alegeți o opțiune validă.");  MotivulCreditarii();  break;  }  } catch (InputMismatchException e) {  System.*out*.println("Opțiune invalidă. Vă rugăm să introduceți un număr.");  in.nextLine();  MotivulCreditarii();  }  }   public String getCodFiscal() {  return codFiscal;  }   public void setCodFiscal(String codFiscal) {  this.codFiscal = codFiscal;  }   public String getDenumire() {  return Denumire;  }   public void setDenumire(String denumire) {  Denumire = denumire;  }   public String getNumeAdministrator() {  return NumeAdministrator;  }   public void setNumeAdministrator(String numeAdministrator) {  NumeAdministrator = numeAdministrator;  }   public String getPersoanaDeContact() {  return PersoanaDeContact;  }   public void setPersoanaDeContact(String persoanaDeContact) {  PersoanaDeContact = persoanaDeContact;  }    private void afișațiMeniul() {  String meniu = """  Alege una din următoarele opțiuni de proprietăți:  1. Proprietăți rezidențiale  2. Proprietăți comerciale  3. Proprietăți industriale  4. Proprietăți pentru uz special  5. Terenuri  6. Acțiuni  7. Obligațiuni  8. Instrumente de piață monetară  9. Fonduri de investiții  10. Derivate financiare  11. Proprietăți necorporale  12. Numerar și echivalente de numerar  Introdu numărul opțiunii dorite:""";   System.*out*.println(meniu);  }  @Override  void CitesteClient() {  try {  System.*out*.print("\033[1;32mIntroduceți codul fiscal:\033[0m ");  setCodFiscal(in.nextLine());   System.*out*.print("\033[1;32mIntroduceți denumirea:\033[0m ");  setDenumire(in.nextLine());   System.*out*.print("\033[1;32mIntroduceți numele administratorului:\033[0m ");  setNumeAdministrator(in.nextLine());   System.*out*.print("\033[1;32mIntroduceți persoana de contact:\033[0m ");  setPersoanaDeContact(in.nextLine());   System.*out*.print("\033[1;32mAlegeți tipul de proprietate din următoarele opțiuni:\033[0m\n");  afișațiMeniul();   int optiune;  try {  optiune = in.nextInt();  in.nextLine();   if (optiune >= 1 && optiune <= *tipuriProprietati*.size()) {  tipProprietate = *tipuriProprietati*.get(optiune - 1);  } else {  System.*out*.println("Opțiune invalidă. Vă rugăm să selectați un număr valid.");  CitesteClient();  return;  }  } catch (InputMismatchException e) {  System.*out*.println("Opțiune invalidă. Vă rugăm să introduceți un număr.");  in.nextLine();  CitesteClient();  return;  }   System.*out*.print("\033[1;32mIntroduceți adresa:\033[0m ");  setAdresa(in.nextLine());   System.*out*.print("\033[1;32mIntroduceți telefonul:\033[0m ");  setTelefon(in.nextLine());   System.*out*.println("\033[1;32mIntroduceți detaliile filialei:\033[0m");  filiala = new Filiala();   MotivulCreditarii();   } catch (Exception e) {  System.*out*.println("A apărut o eroare la citirea datelor. Vă rugăm să încercați din nou.");  e.printStackTrace();  }  }    @Override  public String toString() {  return String.*format*(  "\033[1;31mPersoana Juridica:\033[0m\n" +  "\033[1;33m Cod Fiscal:\033[0m %s\n" +  "\033[1;33m Denumire:\033[0m %s\n" +  "\033[1;33m Nume Administrator:\033[0m %s\n" +  "\033[1;33m Persoana de Contact:\033[0m %s\n" +  "\033[1;33m Adresa:\033[0m %s\n" +  "\033[1;33m Telefon:\033[0m %s\n" +  "\033[1;33m Tip Proprietate:\033[0m %s\n%s",  codFiscal, Denumire, NumeAdministrator, PersoanaDeContact, Adresa, telefon, tipProprietate, super.toString()  );  } } |
| Clasa Test este o aplicație Java care gestionează operațiuni bancare precum adăugarea de clienți (persoane fizice și juridice), afișarea listei de credite, calcularea veniturilor și salvarea datelor într-un fișier. Folosește obiecte serializabile pentru stocarea persistentă a datelor și include interacțiune utilizator prin terminal pentru introducerea informațiilor necesare. | import java.io.\*; import java.util.ArrayList; import java.util.InputMismatchException; import java.util.Scanner;  public class Test {   static Scanner *in* = new Scanner(System.*in*);  static ArrayList<Credit> *lista* = new ArrayList<>();   static void AdaugaInFisier() {  try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("Credite.ser"))) {  out.writeObject(*lista*);  } catch (FileNotFoundException e) {  System.*out*.println("Fisierul nu exista");  } catch (IOException e) {  System.*out*.println("Datele nu au putut fi inscrise in fisier");  }  }   static void DescarcaDinFisier() {  try (ObjectInputStream in = new ObjectInputStream(new FileInputStream("Credite.ser"))) {  *lista* = (ArrayList<Credit>) in.readObject();  } catch (FileNotFoundException e) {  System.*out*.println("Fisierul este gol.");  } catch (IOException e) {  System.*out*.println("Eroare la descarcarea datelor din fisier.");  } catch (ClassNotFoundException e) {  System.*out*.println("Datele din fisier nu coincid cu clasa.");  }  }     static void adaugaPersoanaFizica(Filiala filiala1, Filiala filiala2) {  PersoanaFizica persoanaFizica = new PersoanaFizica();  persoanaFizica.CitesteClient();  persoanaFizica.setFiliala(*selecteazaFiliala*(filiala1, filiala2));  CreditStandard credit = new CreditStandard(persoanaFizica);  credit.CereCredit();  credit.setClient(persoanaFizica);  *lista*.add(credit);  }   static void adaugaPersoanaJuridica(Filiala filiala1, Filiala filiala2) {  PersoanaJuridica persoanaJuridica = new PersoanaJuridica();  persoanaJuridica.CitesteClient();  persoanaJuridica.setFiliala(*selecteazaFiliala*(filiala1, filiala2));  CreditStandard credit = new CreditStandard(persoanaJuridica);  credit.CereCredit();  credit.setClient(persoanaJuridica);  *lista*.add(credit);  }   static Filiala selecteazaFiliala(Filiala filiala1, Filiala filiala2) {  System.*out*.println("\n\033[1;32mAlegeți filiala:\033[0m");  System.*out*.println("1. " + filiala1.getNume());  System.*out*.println("2. " + filiala2.getNume());  int optiune = *in*.nextInt();  *in*.nextLine();  switch (optiune) {  case 1:  return filiala1;  case 2:  return filiala2;  default:  System.*out*.println("Opțiune invalidă. Alegeți o opțiune validă.");  return *selecteazaFiliala*(filiala1, filiala2);  }  }   static void afiseazaListaCredite() {  if (*lista*.isEmpty()) {  System.*out*.println("Nu exista credite inregistrate.");  } else {  for (Credit credit : *lista*) {  System.*out*.println(credit);  }  }  }   static void CalculeazaSalariu(int optiune){  double salariu = 0;  double venit=0;  double profit=0;  double procent=0.07;  for(Credit c: *lista*){  if(!c.getValuta().equalsIgnoreCase("lei"))  c.getContract().setSumaCreditului(c.ConvertireValuta(c.getValuta(), c.getContract().getSumaCreditului()));  salariu+=c.getContract().getSumaCreditului()\*procent;   venit+=c.SumaTotalaReintoarsa(c.contract.getSumaCreditului(), c.getProcentAnual(), c.contract.getTermen());  profit+=c.getContract().getSumaCreditului();  }  profit=venit-profit;  if(optiune==4)  System.*out*.printf("\033[1;34mVentiul angajtului in urma perfectarii contractelor este de %.2f lei\033[0m\n",salariu);  if(optiune==5)  System.*out*.printf("\033[1;34mVenitul curat al bancii in urma acordarii contractelor este de %.2f lei\033[0m\n", profit);  }     public static void main(String[] args) {  *DescarcaDinFisier*();   Filiala filiala1 = new Filiala("Banca de Economii", "Strada Stefan cel Mare, Nr. 1", "022123456");  Filiala filiala2 = new Filiala("Moldindconbank", "Strada Stefan cel Mare, Nr. 2", "022654321");     boolean exit = false;  while (!exit) {  System.*out*.println("\n\033[1;34mMeniu Principal:\033[0m");  System.*out*.println("1. Adauga Persoana Fizica");  System.*out*.println("2. Adauga Persoana Juridica");  System.*out*.println("3. Afiseaza Lista Credite");  System.*out*.println("4. Calculeaza Venit Angajat");  System.*out*.println("5. Calculeaza Venit Banca");  System.*out*.println("6. Salveaza si Iesi");   int optiune;  try {  optiune = *in*.nextInt();  *in*.nextLine();   switch (optiune) {  case 1:  *adaugaPersoanaFizica*(filiala1, filiala2);  break;  case 2:  *adaugaPersoanaJuridica*(filiala1, filiala2);  break;  case 3:  *afiseazaListaCredite*();  break;  case 4:  *CalculeazaSalariu*(4);  break;  case 5:  *CalculeazaSalariu*(5);  break;  case 6:  *AdaugaInFisier*();  exit = true;  break;  default:  System.*out*.println("Opțiune invalidă. Vă rugăm să alegeți o opțiune validă.");  }  } catch (InputMismatchException e) {  System.*out*.println("Opțiune invalidă. Vă rugăm să introduceți un număr.");  *in*.nextLine();  }  }  }   } |

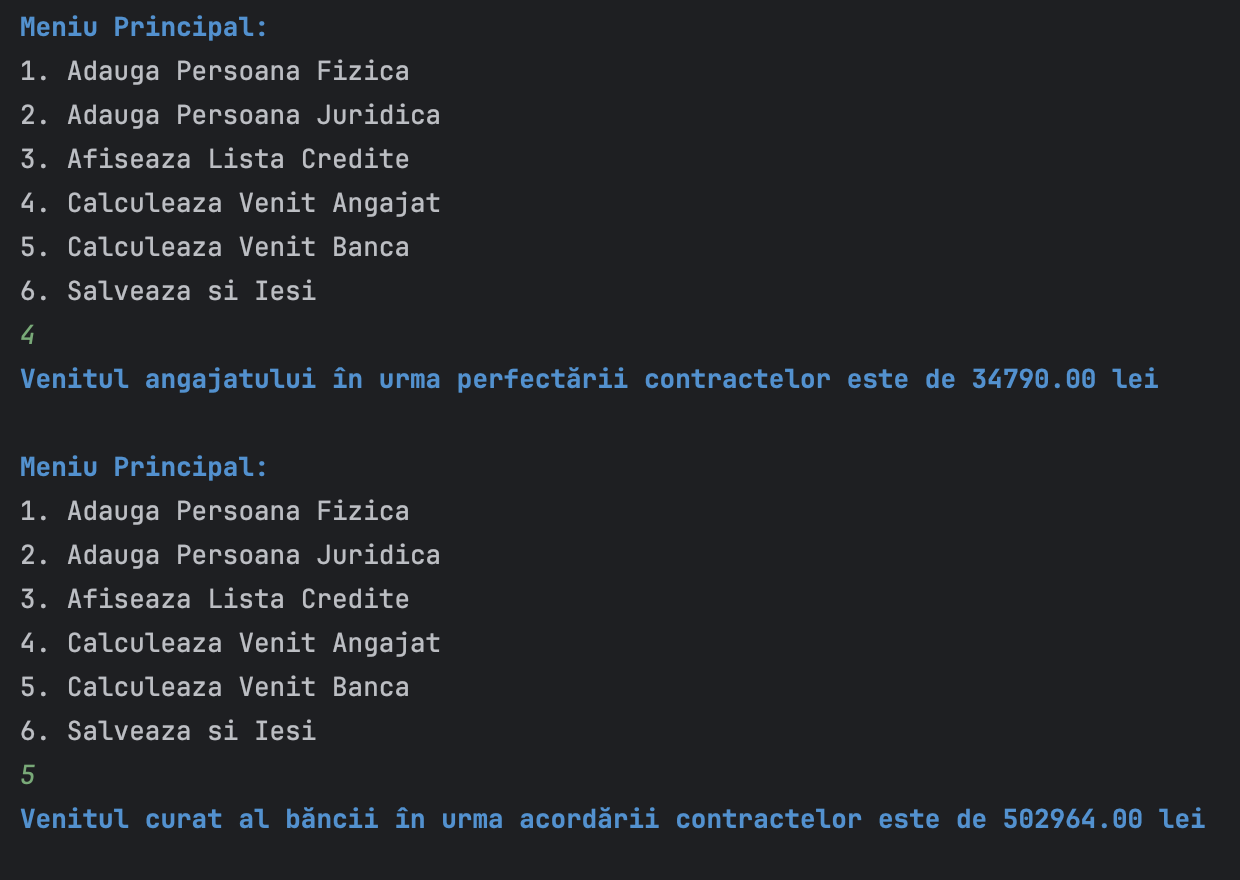
## Rezultatele testării

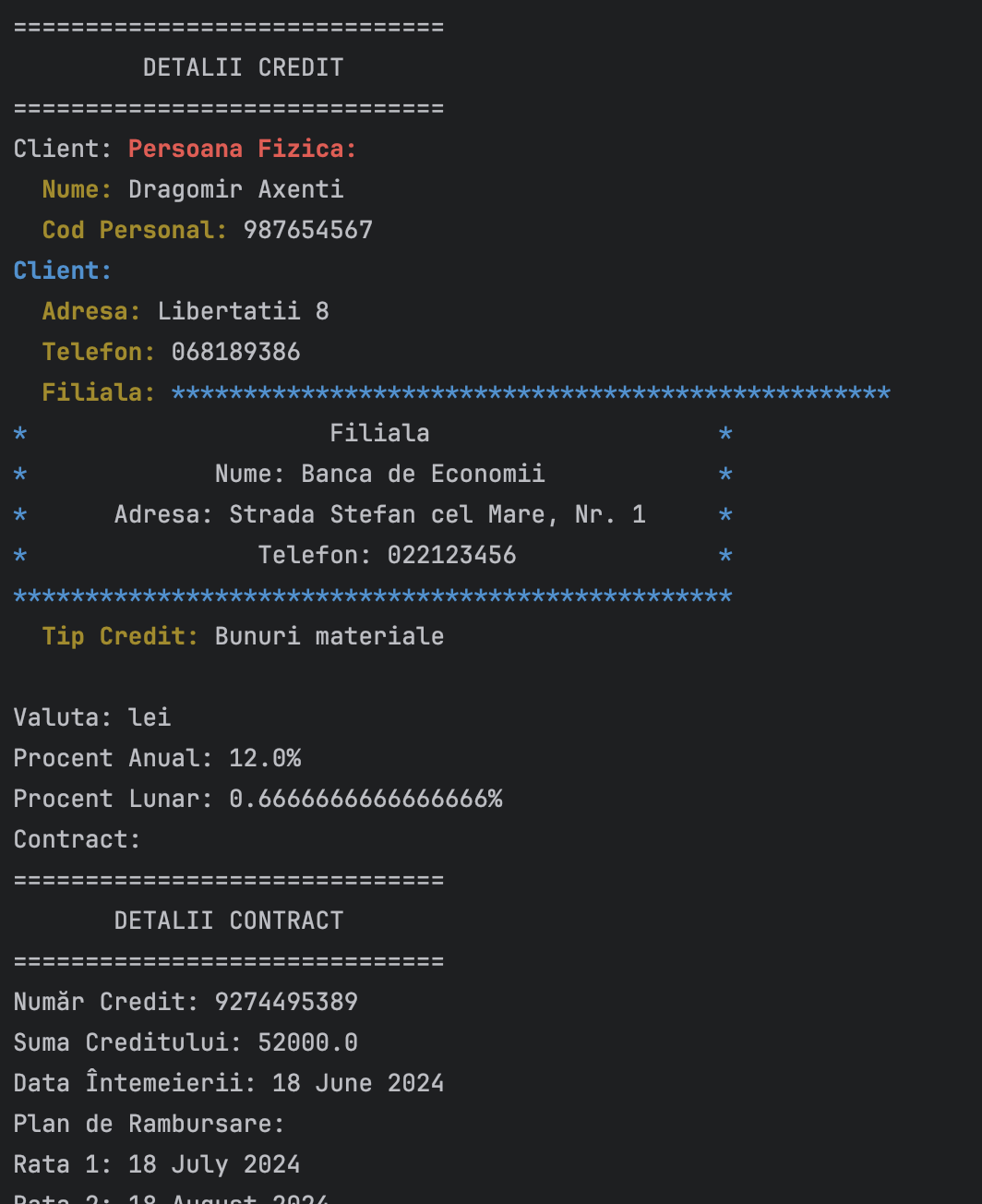
### Date de intrare

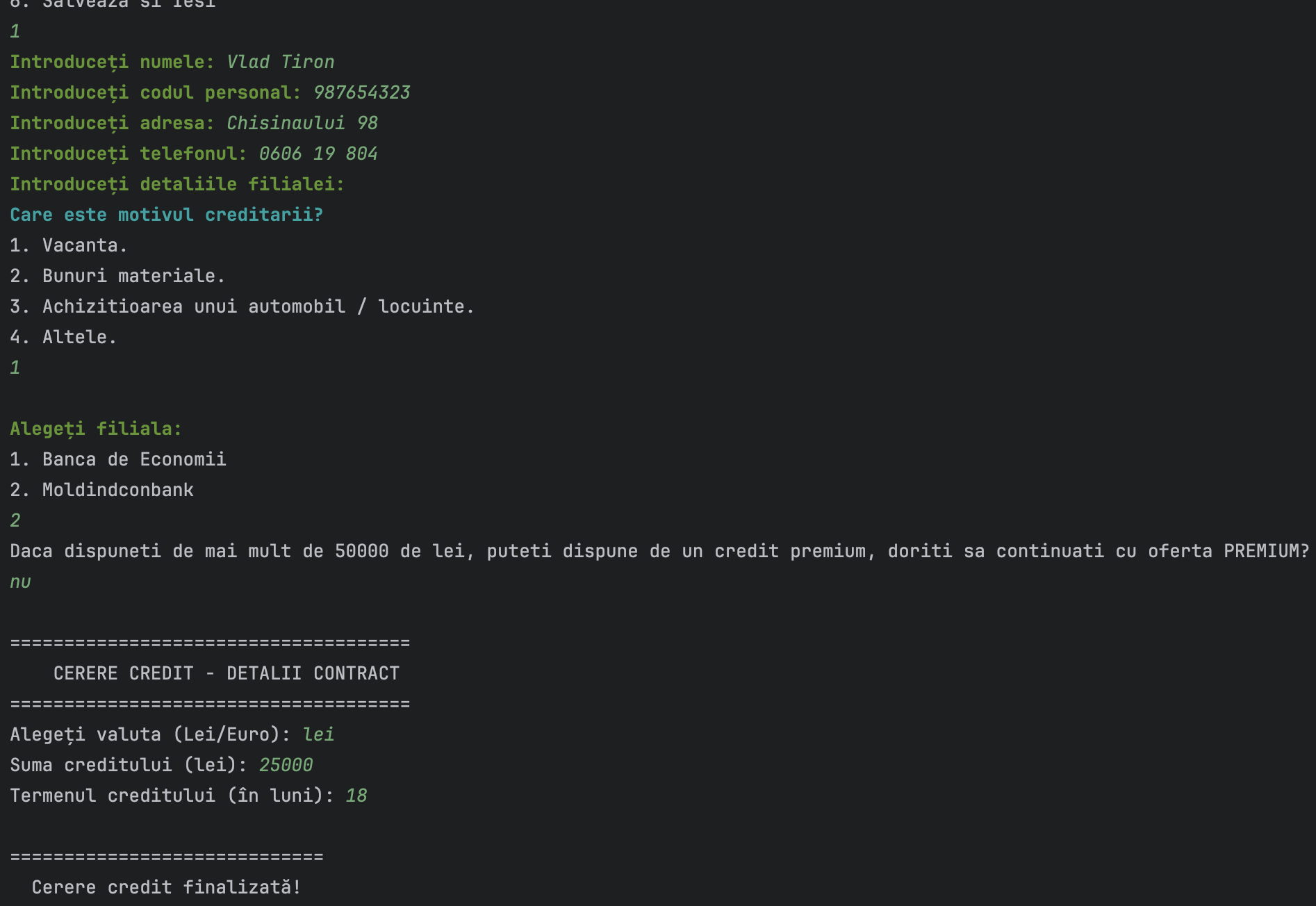
Toate datele sunt salvate în fișierul **„Credite.ser”.**

De la tastatură se vor introduce datele privind identitatea clientului și datele privind specificațiile contractului.

### Testarea funcționalității







# Concluzii

În perioada studiilor mele în programare, am călătorit printr-o serie de provocări și realizări care mi-au adâncit înțelegerea și profesionalismul în C++ și Java. Această călătorie a început cu fundamentul solid al limbajului C++, unde am învățat să manipulez matrice și să gestionez fișiere în diverse aplicații, aprofundându-mă în tehnici de programare. Am descoperit puterea structurilor de date și a algoritmilor în C++, implementând soluții pentru sortarea eficientă a datelor și gestionarea complexității în timpul parcurgerii matricelor în spirala. Această etapă m-a învățat mult despre eficiența codului și importanța alegerii algoritmilor potriviți pentru fiecare problemă.

În timpul trecerii la Java, am descoperit lumea programării orientate pe obiecte într-un mod profund. Am învățat să utilizez interfețele, clasele abstracte și moștenirea pentru a crea aplicații flexibile și ușor de întreținut. Gestionarea excepțiilor și a input/output-ului în Java m-a făcut să înțeleg cum să construiesc aplicații robuste și fiabile, capabile să gestioneze diverse situații neașteptate în mod elegant. Abundența excepțiilor, modurile de tratare a acestora, interfețele și funcționalitățile oferite de acest limbaj de programare ușurează exponențial nivelul de complexitate al scrierii codului, astfel programarea devine o plăcere.

În fiecare etapă a acestui proces, am experimentat cu bucurie provocările și reușitele care m-au ajutat să dezvolt abilități esențiale pentru cariera mea în programare. Capacitatea de a gândi logic și de a rezolva probleme complexe a fost mereu pusă la încercare și îmbunătățită, iar acum sunt pregătit să aplic aceste cunoștințe și experiență în proiecte viitoare care să aducă valoare și inovație în lumea tehnologiei.

# Bibliografie

[https://stackoverflow.com](https://stackoverflow.com/)

[https://github.com](https://github.com/)

<https://www.w3schools.com/java/java_ref_arrays.asp>

[https://www.youtube.com](https://www.youtube.com/)