Universitatea Titu Maiorescu

Facultatea de Informatică, ID

Prezentare Proiect
Metode Avansate de Programare

—

Aplicație Bancară

—

Angeru Florin
Grupa 205

Floroiu Andrei Iulian
Grupa 206

AN 2, 2023-2024

Cuprins

Cuprins

1. Introducere

1.1. Scopul proiectului

Scopul acestui proiect este de a dezvolta o aplicaţie bancară completă, care să permită realizarea unui set variat de operaţiuni bancare esenţiale. Printre aceste operaţiuni se numără crearea de conturi pentru clienţi, gestionarea tranzacţiilor financiare, precum şi generarea de extrase de cont. Aplicaţia este concepută pentru a fi uşor de utilizat şi pentru a oferi o experienţă intuitivă utilizatorilor, facilitând accesul la informaţiile financiare şi gestionarea eficientă a conturilor.

1.2. Cerinţele proiectului

Proiectul trebuie să îndeplinească următoarele cerinţe:

- Gestionarea a cel puţin 8 tipuri diferite de obiecte: Printre acestea se numără clienţi, conturi, tranzacţii, carduri, etc.
- Realizarea a cel puţin 10 acţiuni sau interogări: Inclusiv crearea, modificarea şi ştergerea datelor, precum şi generarea de rapoarte şi interogări complexe.
- Utilizarea colecţiilor Java: Implementarea de colecţii precum ArrayList şi HashMap pentru gestionarea eficientă a datelor.
- Aplicarea conceptelor de moştenire şi polimorfism: Demonstrând astfel abilităţi avansate de programare OOP.
- Implementarea de clase abstracte şi interfeţe: Pentru a defini comportamente comune şi contracte între obiecte.
- Utilizarea de excepţii personalizate şi enumerări: Pentru a gestiona erorile şi a reprezenta constantenţele din aplicaţie.
- Stocarea şi gestionarea datelor în fişiere CSV: Asigurând astfel persistenţa informaţiilor şi posibilitatea de a salva și încărca datele.

2. Funcționalități și cerințe

2.1. Funcţionalităţi implementate

- Crearea şi gestionarea clienţilor: Aplicaţia permite adăugarea de noi clienţi, actualizarea informaţiilor despre clienţi existenţi şi listarea tuturor clienţilor din sistem. De asemenea, permite ștergerea clienţilor care nu mai au conturi active.
- Crearea şi gestionarea conturilor: Utilizatorii pot crea conturi de tip PRIMARY sau SAVINGS pentru clienţi existenţi, modifica detalii despre conturi şi vizualiza lista de conturi. Conturile pot fi șterse doar dacă nu au sold şi nu au carduri asociate.

- Realizarea tranzacţiilor: Aplicaţia permite realizarea de transferuri de fonduri între conturi, retrageri de numerar şi depuneri de fonduri. Toate tranzacţiile sunt logate pentru a asigura trasabilitatea.
- Generarea de extrase de cont: Utilizatorii pot genera extrase de cont pentru un interval de timp specificat, care include toate tranzacţiile realizate în acea perioadă. Extrasele de cont afişează detalii despre fiecare tranzacţie şi soldul final.
- Gestionarea cardurilor: Aplicaţia suportă adăugarea de noi carduri la conturi, blocarea şi deblocarea cardurilor existente, precum şi ştergerea cardurilor.
- Salvarea şi încărcarea datelor din fişiere CSV: Datele sunt salvate automat în fişiere CSV la închiderea aplicaţiei şi sunt încărcate din aceste fişiere la pornirea aplicaţiei pentru a asigura persistenţa datelor.
- Logarea acţiunilor: Fiecare acţiune realizată în aplicaţie este logată într-un fişier CSV pentru a asigura trasabilitatea şi auditul operaţiunilor. Acest lucru este esenţial pentru securitate şi pentru a răspunde eventualelor dispute.

2.2. Cerinţe tehnice

- Limbaj de programare: Java
- Framework de logare: Log4j
- Stocarea datelor: Fişiere CSV
- IDE recomandate: IntelliJ IDEA, Eclipse
- Instrumente de compilare şi rulare: Gradle sau Maven

3. Arhitectura proiectului

3.1. Structura proiectului

- database: Directorul conţine fişierele CSV pentru stocarea datelor, inclusiv:
  - accounts.csv
  - audit.csv
  - cards.csv
  - customers.csv
  - transactions.csv
- src/main/java/com/banking: Directorul principal al codului sursă, organizat în mai multe pachete:
  - enums: Pachet care conţine enumerările pentru tipurile de conturi şi tranzacţii (AccountType.java, TransactionType.java).
  - exceptions: Pachet care conţine excepţiile personalizate (InsufficientFundsException.java, InvalidAccountException.java).
  - models: Pachet care defineşte clasele model pentru entităţile aplicaţiei (Account.java, BankStatement.java, Card.java, Customer.java, PrimaryAccount.java, SavingsAccount.java, Transaction.java).

- services: Pachet care conţine serviciile pentru gestionarea logicii aplicaţiei (AuditService.java, BankService.java, BankServiceInterface.java, StorageService.java).
- Main.java: Clasa principală care iniţiază aplicaţia şi conţine logica meniului.

4. Descrierea componentelor

4.1. Main.java

Clasa principală care conţine meniul interactiv al aplicaţiei şi metodele pentru fiecare acţiune disponibilă în meniu.

Acţiunile disponibile includ:

- Crearea şi editarea clienţilor şi conturilor.
- Realizarea tranzacţiilor financiare (transferuri, retrageri, depuneri).
- Generarea de extrase de cont.
- Gestionarea cardurilor (adăugare, blocare, deblocare).
- Ieşirea din aplicaţie.

Exemple de metode:

- createCustomer(): Permite utilizatorului să introducă informaţiile necesare pentru a crea un nou client. Această metodă colectează detalii precum numele, prenumele şi vârsta clientului.
- createAccount(): Permite utilizatorului să creeze un cont pentru un client existent, selectând tipul contului (PRIMARY sau SAVINGS). Metoda validează ID-ul clientului şi tipul contului înainte de a crea contul.
- makeTransaction(): Permite utilizatorului să efectueze o tranzacţie între conturi. Utilizatorul poate selecta tipul de tranzacţie (transfer, retragere, depunere) şi introduce suma dorită.
- generateBankStatement(): Permite utilizatorului să genereze un extras de cont pentru o perioadă specificată. Metoda solicită utilizatorului să introducă ID-ul contului şi intervalul de timp dorit.
- listCustomers(): Afişează o listă cu toţi clienţii existenţi în sistem, inclusiv detalii despre fiecare client, precum ID-ul, numele şi numărul de conturi deţinute.
- listAccounts(): Afişează o listă cu toate conturile existente, inclusiv detalii despre soldul fiecărui cont şi tipul acestuia.
- editCustomer(): Permite utilizatorului să modifice informaţiile unui client existent. Utilizatorul poate edita numele, prenumele şi vârsta clientului.
- editAccounts(): Permite utilizatorului să modifice detalii despre conturile existente. Utilizatorul poate schimba rata dobânzii pentru conturile de economii şi poate adăuga sau şterge carduri pentru conturile primare.

4.2. Account.java

Clasă abstractă care defineşte atribute şi metode comune pentru toate tipurile de conturi bancare.

Atributele includ:

- accountId: ID-ul unic al contului.
- balance: Soldul contului.
- owner: Clientul care deţine contul.
- transactions: Lista tranzacţiilor efectuate pe acest cont.
- cards: Lista cardurilor asociate contului.

Metode comune:

- deposit(double amount): Metodă abstractă pentru depunerea de fonduri în cont.
- withdraw(double amount): Metodă abstractă pentru retragerea de fonduri din cont.
- addCard(Card card): Adaugă un card la cont.
- removeCard(Card card): Șterge un card din cont.

4.3. BankStatement.java

Clasă care reprezintă un extras de cont.

Aceasta conţine:

- statementId: ID-ul unic al extrasului de cont.
- account: Referinţa la contul pentru care este generat extrasul.
- startDate şi endDate: Intervalul de timp pentru care este generat extrasul.
- transactions: Lista tranzacţiilor realizate în acest interval.
- closingBalance: Soldul final al contului la sfârşitul perioadei.

Metode:

- toString(): Metodă care returnează o reprezentare textuală a extrasului de cont, incluzând detalii despre tranzacţiile efectuate și soldul final.

4.4. Card.java

Clasă care reprezintă un card bancar.

Atributele includ:

- cardNumber: Numărul cardului.
- expirationDate: Data de expirare a cardului.
- account: Contul asociat cardului.
- blocked: Starea cardului (blocat sau deblocat).

Metode:

- block(): Blochează cardul.
- unblock(): Deblochează cardul.

- toString(): Metodă care returnează o reprezentare textuală a cardului, incluzând starea și data de expirare.

4.5. Customer.java

Clasă care reprezintă un client bancar.

Atributele includ:

- id: ID-ul unic al clientului.
- name: Numele clientului.
- surname: Prenumele clientului.
- age: Vârsta clientului.
- accounts: Lista conturilor deținute de client.

Metode:

- addAccount(Account account): Adaugă un cont la lista de conturi ale clientului.
- toString(): Metodă care returnează o reprezentare textuală a clientului, incluzând ID-ul, numele și conturile deținute.

4.6. PrimaryAccount.java și SavingsAccount.java

Clase care extind clasa abstractă Account și implementează funcționalitățile specifice pentru conturile de tip PRIMARY și SAVINGS.

4.6.1. PrimaryAccount.java

Metode specifice:

- deposit(double amount): Adaugă suma specificată la soldul contului și înregistrează tranzacția.
- withdraw(double amount): Scade suma specificată din soldul contului și înregistrează tranzacția. Aruncă InsufficientFundsException dacă fondurile sunt insuficiente.

4.6.2. SavingsAccount.java

Atribut specific: interestRate (rata dobânzii)

Metode specifice:

- deposit(double amount): Adaugă suma specificată la soldul contului și înregistrează tranzacția.
- withdraw(double amount): Scade suma specificată din soldul contului și înregistrează tranzacția. Aruncă InsufficientFundsException dacă fondurile sunt insuficiente.
- applyInterest(): Calculează și adaugă dobânda la soldul contului și înregistrează tranzacția.

4.7. Transaction.java

Clasă care reprezintă o tranzacţie bancară.

Atributele includ:

- transactionId: ID-ul unic al tranzacţiei.
- timestamp: Data şi ora la care a avut loc tranzacţia.
- amount: Suma tranzacţionată.
- type: Tipul tranzacţiei (depunere, retragere, transfer).
- account: Contul asociat tranzacţiei.

Metode:

- toString(): Metodă care returnează o reprezentare textuală a tranzacţiei, incluzând detalii despre cont, suma şi tipul tranzacţiei.

4.8. AccountType.java şi TransactionType.java

Enumerări pentru tipuri de conturi şi tranzacţii.

AccountType.java - Enumerare pentru tipurile de conturi (PRIMARY, SAVINGS).

TransactionType.java - Enumerare pentru tipurile de tranzacţii (DEPOSIT, WITHDRAWAL, TRANSFER).

4.9. InsufficientFundsException.java şi InvalidAccountException.java

Excepţii personalizate.

InsufficientFundsException.java - Excepţie aruncată atunci când fondurile din cont sunt insuficiente pentru a efectua o retragere sau un transfer.

InvalidAccountException.java - Excepţie aruncată atunci când se încearcă accesarea unui cont invalid sau inexistent.

4.10. AuditService.java

Serviciu pentru logarea(inregistrarea) acţiunilor. Utilizează Log4j pentru logarea(inregistrarea) activităţilor în fişierul audit.csv.

Metode:

- logAction(String actionName): Loghează(inregistreaza) acţiunea specificată în fişierul de audit, împreună cu un timestamp.

4.11. StorageService.java

Serviciu pentru gestionarea stocării datelor în fișiere CSV. Utilizează singleton pattern pentru a asigura existența unei singure instanțe a serviciului.

Metode:

- loadCustomers(): Încarcă lista de clienți din fișierul customers.csv.
- saveCustomers(Collection<Customer> customers): Salvează lista de clienți în fișierul customers.csv.
- loadAccounts(Map<String, Customer> customerMap): Încarcă lista de conturi din fișierul accounts.csv.
- saveAccounts(Collection<Account> accounts): Salvează lista de conturi în fișierul accounts.csv.
- loadTransactions(Map<String, Account> accountMap): Încarcă lista de tranzacții din fișierul transactions.csv.
- saveTransactions(Collection<Transaction> transactions): Salvează lista de tranzacții în fișierul transactions.csv.
- loadCards(Map<String, Account> accountMap): Încarcă lista de carduri din fișierul cards.csv.
- saveCards(Collection<Card> cards): Salvează lista de carduri în fișierul cards.csv.

5. Utilizarea aplicației

5.1. Instrucțiuni pentru rularea aplicației:

5.1.1. Clonarea proiectului:

- Clonează proiectul din repository-ul său.
- Navighează la directorul proiectului în terminal.

```
git clone <repository-url>
cd <project-directory>
```

5.1.2. Compilarea şi rularea proiectului:

Utilizează un IDE Java (de ex. IntelliJ IDEA, Eclipse) sau din linia de comandă folosind Gradle.

```
gradle build
gradle run
```

5.2. Exemple de scenarii de utilizare:

- Crearea unui nou client:
  - Selectează opţiunea 1 din meniul principal.
  - Introdu numele, prenumele şi vârsta clientului.
  - Clientul este adăugat şi confirmarea apare pe ecran.

- Crearea unui nou cont pentru un client existent:
  - Selectează opţiunea 2 din meniul principal.
  - Introdu ID-ul clientului şi tipul contului (PRIMARY sau SAVINGS).
  - Contul este creat şi confirmarea apare pe ecran.

- Efectuarea unei depuneri într-un cont:
  - Selectează opţiunea 3 din meniul principal.
  - Introdu ID-ul clientului şi selectează contul dorit.
  - Selectează opţiunea pentru depunere şi introdu suma.
  - Suma este depusă şi confirmarea apare pe ecran.

- Generarea unui extras de cont:
  - Selectează opţiunea 4 din meniul principal.
  - Introdu ID-ul contului şi perioada dorită (data de început şi de sfârşit).
  - Extrasul de cont este generat şi afişat pe ecran.
- Transfer între două conturi:
  - Selectează opţiunea 5 din meniul principal.
  - Introdu ID-urile conturilor sursă şi destinaţie.
  - Introdu suma de transferat.
  - Confirmă transferul şi verifică soldurile actualizate.

6. Testare și validare

6.1. Metodologia de testare

Proiectul a fost testat prin rularea unor scenarii de utilizare care acoperă toate funcţionalităţile principale. Testele au fost realizate manual, iar logurile de audit au fost verificate pentru a asigura corectitudinea operaţiunilor. Testele au fost realizate în mai multe etape:

6.1.1. Testare funcţională:

Verificarea corectitudinii fiecărei funcţionalităţi implementate (crearea de clienţi, conturi, tranzacţii etc.).

Asigurarea că toate operaţiunile returnează rezultatele aşteptate.

6.1.2. Testare de integrare:

Verificarea interacţiunii dintre diferitele componente ale sistemului (de exemplu, între modulul de clienţi şi modulul de conturi).

Asigurarea că datele sunt corect partajate între module.

6.1.3. Testare de regresie:

Ne asiguram că modificările recente în cod nu au introdus erori în funcţionalităţile existente.

6.1.4. Rezultatele testelor

Toate funcţionalităţile au fost testate şi au funcţionat corect. Nu au fost identificate erori majore. Exemple de acţiuni testate:

Crearea şi editarea clienţilor şi conturilor: Toate datele introduse au fost corect salvate şi încărcate din fişierele CSV.

Realizarea de tranzacţii (transferuri, retrageri, depuneri): Tranzacţiile au fost corect înregistrate în soldurile conturilor şi în logurile de audit.

Generarea de extrase de cont: Extrasele au fost corect generate, incluzând toate tranzacţiile din intervalul specificat.

Gestionarea cardurilor (adăugare, blocare, deblocare): Toate operaţiunile ce implica carduri au fost corect realizate şi înregistrate în starea conturilor.