Universitatea Tehnică de Construcții din București
Facultatea de Hidrotehnică
Specializarea: Automatică și Informatică Aplicată

Proiect Programarea calculatoarelor si limbaje de programare III

Profesor coordonator:
Ș.l. univ. dr. ing. Ramona - Oana Flangea

Student:
Lavinia-Teodora Gheorghe

București, 2026

Universitatea Tehnică de Construcții din București
Facultatea de Hidrotehnică
Specializarea: Automatică și Informatică Aplicată

SISTEM DE GESTIONARE A BIBLIOTECII ONLINE THE BOOK NOOK

Profesor coordonator:
Ș.I. univ. dr. ing. Ramona - Oana Flangea

Student:
Lavinia-Teodora Gheorghe

CUPRINS

I. INTRODUCERE

1.1. MOTIVAREA ALEGERII TEMEI

Alegerea de a realiza un sistem de gestionare a unei biblioteci personale provine din pasiunea pentru lectură. De-a lungul timpului am adunat un număr mare de cărți acasă, în biblioteca fizica personală, având o varietate de genuri literare. Din acest motiv, organizarea cărților a devenit o necesitate pentru mine și am ajuns să testez o varietate de aplicații de pe piață. Cu toate acestea, la fiecare aplicație în parte a lipsit câte un aspect. Acest proiect mi-a oferit ocazia de a realiza sistemul de gestionare conform dorințelor și nevoilor proprii, sistem care în timp ar putea fi dezvoltat într-o aplicație complexă și disponibilă nu doar pentru mine, dar și pentru ceilalți iubitori de lectură.

1.2. OBIECTIVELE PROPUSE ÎN CADRUL LUCRĂRII

În cadrul acestui proiect, mi-am creat o listă de obiective pentru a ajunge la rezultatul mult dorit. Scopul principal al proiectului este realizarea unei aplicații care se ocupă cu gestionarea eficientă a cărților dintr-o bibliotecă personală, care să ofere utilizatorilor un mod simplu și prietenos pentru organizarea cărților. Primul obiectiv este crearea unei opțiuni pentru utilizator de a adăuga în sistem detalii referitoare la o carte, cum ar fi cele generale, titlu și autor, dar și câteva mai speciale, precum formatul cărții, având în vedere popularizarea formatelor digitale din ultima vreme. Totodată, se pot clasifica cărțile și in funcție de statusul lor, citită sau pe lista de dorințe. În acest mod, biblioteca devine mai organizată, având mai multe criterii de orientare.

Alt obiectiv personal este oferirea posibilității de a sorta cărțile după un criteriu ales. Câteva opțiuni de sortare sunt după titlu, după formatul cărții, după rating-ul oferit, dar și altele. Astfel, utilizatorul poate vizualiza mai eficient cărțile pe care le deține, după filtrul dorit.

Pe deasupra, aplicația oferă posibilitatea de a căuta o carte. În urma acestei căutări sunt afișate detaliile cărții, într-un mod cât mai lizibil și prietenos. Altă trăsătură a sistemului este de a edita informațiile deja înregistrate în baza de date, sau ulterior, ștergea completă a unei cărți.

Un ultim obiectiv este crearea unei interfețe intuitive și ușor de folosit, care să ofere acces rapid la date și, totodată, o experiență plăcută utilizatorului.

1.3. STRUCTURA LUCRĂRII

Lucrarea este compusă din trei capitole, plus un capitol dedicat concluziilor și perspectivelor de dezvoltare viitoare. Proiectul prezintă procesul de dezvoltare al unei aplicații foarte folositoare pentru iubitorii de lectură.

- **Capitolul I:**

În cadrul primului capitol, mi-am exprimat obiectivele propuse în legătură cu prezenta lucrare, dar și o argumentație solidă privind oportunitatea de a aborda această temă.

- **Capitolul II**:

În al doilea capitol, am pus accent pe tehnologiile și limbajul de programare folosite în realizarea acestui proiect, dar și pe elementele specifice limbajului. Pe scurt, am utilizat limbajul de programare Java în cadrul aplicației Eclipse IDE for Java Developers.

- **Capitolul III:**

În cel de-al treilea capitol, am detaliat pas cu pas codul din spatele aplicației, modul în care se utilizează și am oferit exemple din rularea acesteia.

Lucrarea în ansamblu își propune să ofere o înțelegere aprofundată a etapelor de dezvoltare a bibliotecii online, să consolideze cunoștințele dobândite și să demonstreze aplicabilitatea limbajului de programare Java în crearea unui proiect funcțional.

II. TEHNOLOGII SI INSTRUMENTE HARDWARE SI SOFTWARE FOLOSITE IN IMPLEMENTAREA PROIECTULUI

II.1. LIMBAJUL DE PROGRAMARE JAVA

Java este un limbaj de programare OOP sau orientat-obiect, dezvoltat de James Gosling la Sun Microsystems (acum filială Oracle), la începutul anilor ’90 și lansat în 1995.

După aproape trei decenii, acest limbaj orientat-obiect (OOP), mai simplu decât limbajele profesionale C si C++, continua sa domine aplicațiile software, pentru ca este flexibil, eficient si ușor de înțeles.

Java este un program care poate fi folosit la realizarea unei game largi de aplicații software. După ce un programator dezvoltă o aplicație Java, aceasta poate rula pe majoritatea sistemelor de operare (OS), incluzând Windows, Linux și Mac OS. Java este un limbaj versatil, lucru care a contribuit în destul de mare măsură la succesul său.

Este important de menționat că Java poate fi folosit pentru a dezvolta aplicații complete care pot rula pe un singur computer sau care pot fi distribuite între servere și clienți într-o rețea. De asemenea, poate fi folosit pentru a programa mini aplicații sau applets care nu sunt independente, ci sunt parte a unei pagini web și facilitează interacțiunea utilizatorului cu interfața grafică (intrând în competiție cu alte produse de acest tip, precum Adobe Flash sau Microsoft Silverlight).

Programele Java se folosesc pe desktop-uri, servere, smartphone-uri, card-uri sau discuri Blu-ray (BD).

II.1.1. TEHNOLOGIA JAVA ȘI COMPONENTELE SALE

- In tehnologia Java sunt cuprinse patru componente:

    1. Mașina virtuală, denumită JVM (Java Virtual Machine) care interpretează fișierele cu extensia .class ce conține “bytecode”.

“Bytecode” reprezintă cod compilat care este procesat de un program, denumit mașina virtuală, și nu cod executabil care rulează pe mașina reală ce dispune de un procesor hardware.

“Bytecode” este format din instrucțiuni generice care sunt convertite (transformate) de mașina virtuală în instrucțiuni specifice procesorului mașinii reale. De aceea se spune că aplicațiile Java sunt independente de platformă (hardware și software).

2. Limbajul Java propriu-zis. Limbajul Java este orientat pe obiecte și se aseamănă din punct de vedere al sintaxei cu C++.
3. Un compilator care generează fișiere cu extensia .class. Compilatorul Java traduce instrucțiunile scrise în limbajul Java (stocate în fișiere cu extensia .java) în instrucțiuni generice "bytecode"(stocate în fișiere cu extensia .class) care sunt "înțelese" de mașina virtuală.
4. Biblioteca de clase Java, denumită și Java API. Această bibliotecă conține un set de componente utile care pot fi reutilizate de programatori în diverse aplicații informatice.

- Implementări ale limbajului Java

Cele mai importante implementări ale limbajului Java sunt realizate de firmele Sun și IBM.

Microsoft a creat propria sa implementare Java, disponibilă numai pe platforme Windows.

- Utilizarea claselor de obiecte din pachetele predefinite (pachete API) Java

Pot exista unele operații care sunt frecvent utilizate în cadrul unei aplicații Java (cum ar fi: citirea datelor, scrierea datelor, extragerea radicalului, desenarea de obiecte grafice etc) pentru care nu există instrucțiuni specifice ale limbajului Java.

Din acest motiv, s-au construit pachete predefinite de clase de obiecte (pachete API) (elaborate de firma proiectantă a mediul de programare folosit) care conțin colecții de clase de obiecte de utilitate generală grupate pe categorii.

Mediul de programare J 2 SDK oferă peste 70 de pachete predefinite, printre care, mai des utilizate sunt:

- java.lang care oferă clase fundamentale pentru limbajul Java: clasa Integer, clasa Math, clasa System, clasa String etc;
- java.io care oferă modalități de citire/scriere a datelor prin intermediul fluxurilor de date, a fișierelor etc;
- java.util care conține clase pentru utilizarea colecțiilor de date abstracte de tip stivă și coadă, pentru manevrarea datelor calendaristice și a timpului;
- java.util.jar pentru citirea/scrierea fisierelor in format jar (Java Archive);
- java.math care oferă clase specializate in calcul matematic;

- java.text care oferă clase pentru manevrarea textului, a datelor calendaristice, a timpului și a mesajelor într-o manieră independentă de limba utilizată;
- java.net care pune la dispoziție clase pentru implementarea aplicațiilor de rețea.

II.1.2. FIȘIERE

Un fișier este o colecție de date indicat printr-un nume și o extensie. Numele este despărțit de extensie prin punct. Există doua tipuri de fișiere: fișiere text și fișiere binare. Un fișier text conține text (cifre si caractere). Un fișier binar poate conține si imagini, baze de date, etc.
Pentru a lucra cu fișiere în Java, se creează un obiect de tip File și se deschide un flux de date. În Java, fluxurile sunt de doua tipuri: cele pentru text și cele pentru date binare.

Exemple de operațiuni cu fișiere:

- Scriere în fișier:
  - FileWriter: fluxul de bază pentru scrierea caracterelor în fișier.
  - BufferedWriter: stochează datele din buffer înainte de a le trimite pe disc.
- Citirea din fișier:
  - FileReader: fluxul de bază pentru citirea caracterelor.
  - BufferedReader: permite citirea textului linie cu linie
  - Scanner: O utilizate flexibilă care poate fi asociată unui fișier pentru a citi cuvinte, numere sau linii întregi.

Pentru proiectul meu am folosit doar fluxuri pentru text și am creat un fișier “carti.txt” pentru a salva datele introduse de utilizatori, dar și pentru a putea lucra cât mai simplu și structurat.

II.1.3. EXCEPȚII JAVA

Excepțiile în Java sunt evenimente care apar în timpul execuției programului care perturbă fluxul normal de instrucțiuni. Ele reprezintă un mecanism de tratare a erorilor care permite programatorilor să gestioneze și să răspundă unor condiții neașteptate sau problematice într-un mod organizat și controlat.
Structura:

- Throwable (clasa de bază)
  - Error (erori grave, de obicei nu le tratăm)
  - Exception (excepții pe care le putem trata)
    - RuntimeException
    - Alte excepții (checked exceptions)

Tipuri de excepții:

- Excepții verificate (IOException, SQLException, ClassNotFoundException)
- Excepții neverificate (NullPointerException, ArrayIndexOutOfBoundsException, ArithmeticException)
- Erori (StackOverFlowError, OutOfMemoryError)
- Excepții personalizate (Acestea extind Exception, RunTimeException)

II.1.4. BLOCUL TRY...CATCH

Administrarea unei excepții se face într-un bloc *try-catch*. Instrucțiunile care pot genera excepții (apeluri de metode sau instrucțiuni *throw*) se plasează într-un bloc try după care apar unul sau mai multe blocuri catch care administrează efectiv fiecare tip de excepție care poate fi generată.

II.1.5. INSTRUCȚIUNEA SWITCH CASE

De multe ori în codurile noastre o să avem nevoie de o modalitate de a executa anumite linii de cod doar dacă se îndeplinește o anumită condiție. Pentru a rezolva această necesitate, s-au creat structurile de decizie, deseori numite și structuri alternative.

În plus față de instrucțiunea de decizie foarte utilizată, *if else*, am ales să folosesc și instrucțiunea *switch case*. Aceasta este foarte utilă în cazul în care vrem să comparăm o valoare cu mai multe alte valori.

II.1.6. PROGRAMAREA ORIENTATĂ PE OBIECTE (POO)

Există diferite abordări ale dezvoltării software. Una dintre cele mai populare și mai eficiente este programarea orientată pe obiecte (POO). Cu ajutorul acesteia se pot crea, extinde și întreține proiecte destul de complexe.

Structura POO este alcătuită din obiecte, clase, atribute și metode:

- Obiecte – instanțe ale unor clase care reprezintă entități reale sau abstracte.
- Clase – șabloane pentru crearea obiectelor care definesc atributele și metodele acestora.
- Atribute – date despre obiect care stochează starea acestuia.
- Metode – funcții care pot schimba starea obiectului sau pot efectua anumite acțiuni.

În Java un obiect este o variabilă complexă care se caracterizează prin:

- o structură, descrisă de atributele (proprietățile) sale;
- o stare, descrisă de valorile pe care le iau la un moment dat atributele sale;
- un set de operații prin intermediul cărora se poate manevra (accesa sau modifica) starea sa.

POO se bazează pe patru principii principale: încapsulare, moștenire și polimorfism. Să analizăm mai îndeaproape pe fiecare dintre ele.

Încapsularea înseamnă gruparea datelor și a operațiilor asupra acestor date în același întreg (agregat) având grijă să se ascundă detaliile de implementare (proiectare-realizare) ale acestui întreg. Deci, datele sunt "ascunse", iar accesul la aceste date se realizează numai prin intermediul metodelor încapsulate cu ele.

Prin moștenire, o clasă nouă dobândește imediat tot comportamentul unei clase existente. Această clasă nouă se numește clasă derivată din clasa existentă. O clasă de obiecte derivată dintr-o altă clasă existentă păstrează toate proprietățile și metodele acesteia din urmă aducând, în plus, proprietăți și metode noi.

Polimorfismul reprezintă capacitatea unui obiect de a apărea sub diferite forme. În Java, polimorfismul înseamnă că o singură variabilă referință de tipul unei superclase poate fi folosită pentru a referi mai multe obiecte (instanțe) din clase derivate direct sau indirect din aceeași superclasă, în diferite momente ale execuției unui program.

În Java există trei modificatori de acces ai metodelor unei clase:

- modificatorul *public*;
- modificatorul *protected*;
- modificatorul *private*.

O metoda constructor este o metoda apelată atunci când obiectul este creat si inițializat, folosind operatorul new. Spre deosebire de alte metode, o metodă constructor nu poate fi apelată direct în cadrul programului; Java apelează metodele constructor în mod automat.

Un obiect de tip Class este o instanță a clasei Class, existentă în pachetul java.lang. Uneori clasa Class mai este denumită si meta clasă. Mașina virtuală Java atașează fiecărei clase, existente într-un program, un obiect de tipul Class, care este utilizat când se creează instanțe ale clasei respective. Acest obiect de tip Class poate fi utilizat si de programator pentru a afla informații despre o clasa oarecare folosita in program.

Metoda *getClass()* returnează un obiect de tip Class care reprezintă clasa din care face parte obiectul respectiv (referit printr-o variabila referință).

O metodă des folosită din clasa Class este *getName()* care permite obținerea numelui unei clase.

Pentru orice clasă putem scrie o serie de metode speciale: metode de tip *setter* prin care atribuim indirect valori atributelor si metode de tip *getter* prin care extragem valori ale atributelor.

II.2. MEDIUL DE DEZVOLTARE ECLIPSE

În informatică, Ecplise este un program de dezvoltare integrat pentru dezvoltarea diverselor aplicații informatice, utilizând în speical limbajul Java, precum și alte limbaje, inclusiv C/C++, Python, PERL, Ruby și multe altele.

Fiind gratuit și open-source, Ecplise IDE este unul dintre cele mai populare IDE JAVA de pe piață. Ecosistemul său de plug-in-uri îl face îndrăgit de dezvoltatori, deoarece accepta plug-in-uri și funcții personalizabile pentru orice aplicație.

Ecplise IDE poate rula pe cele mai populare sisteme de operare, inclusiv Windows, Mac OS și Linux.

Eclipse oferă un spațiu de lucru în care toate proiectele pot fi combinate într-unul singur. Fișierele sursă, artefactele și imaginile pot fi stocate în acest spațiu de lucru unificat. Acest spațiu de lucru poate fi accesat din bara de instrumente sau din cea de meniu. Astfel, fișierele pot fi organizate în mod convenabil.

III. STUDIU DE CAZ – THE BOOK NOOK

III.1. IMPLEMENTARE PAS CU PAS

În primul rând, codul începe cu apelarea unui pachet Java, care cuprinde tot proiectul. Un pachet Java reprezintă o entitate logică care permite organizarea claselor și gestionarea spațiului de nume. Astfel se pot grupa clase și interfețe care au legătură între ele. Prin folosirea pachetelor se pot evita conflictele de nume între clase, fiind permisă existența în cadrul aceleași aplicații a mai multe clase având același nume dar organizate în pachete diferite.

Mai apoi, am importat pachetul fundamental *java.io.**, care oferă o gamă completă de clase și interfețe necesare pentru a efectua operațiuni de intrare/ieșire (I/O), cum ar fi citirea și scrierea de date în și din fișiere, manipularea fluxurilor de date (streams), și comunicarea prin canale, oferind astfel funcționalități esențiale pentru dezvoltarea aplicațiilor care necesită manipularea eficientă și sigură a datelor.

Am creat două clase de excepții, una pentru cazul în care numărul de informații ale unei cărți excede limitele stabilite, și una pentru atunci când biblioteca este încărcată la maxim cu cărți.

```
1  package sistem_gestionare_biblioteca;
2
3  import java.io.*;
5
6
7  class IncorrectNrofDetailsException extends Exception {
8      public IncorrectNrofDetailsException(String msg) {
9          super(msg);
10     }
11 }
12
13 class LibraryFullException extends Exception {
14     public LibraryFullException(String msg) {
15         super(msg);
16     }
17 }
18
```

M-am folosit de două clase importante, *class Book{}* și *class Library{}* pentru a gestiona mai eficient codul. Clasa Book cuprinde atributele specifice unei cărți, dar și metodele de tip getter si setter pentru fiecare atribut. În această clasă, am adăugat și un constructor Book() fără parametri, unde am setat valoarea inițială pentru atribute.

De asemenea, am creat și o funcție de tip *String* pentru a converti mai ușor toate datele în formatul din fișier.

```
20  class Book {
21
22      private String title;
23      private String author;
24      private String category;
25      private String format;
26      private String status;
27      private int nr_pages;
28      private int publication_year;
29      private int rating;
30
31      public Book() {
32          this.title = "";
33          this.author = "";
34          this.category = "";
35          this.format = "";
36          this.status = "";
37          this.nr_pages = 0;
38          this.publication_year = 0;
39          this.rating = -1;
40      }
41
```

```
111     public String toFileFormat() {
112         if(rating  == -1) {
113             return title + ";" +
114                         author + ";" +
115                         category + ";" +
116                         format + ";" +
117                         status + ";" +
118                         nr_pages + ";" +
119                         publication_year;
120         } else {
121             return title + ";" +
122                         author + ";" +
123                         category + ";" +
124                         format + ";" +
125                         status + ";" +
126                         nr_pages + ";" +
127                         publication_year + ";" +
128                         rating;
129         }
130     }
131 }
```

În a doua clasă, Library{} am șapte metode specifice fiecărei opțiuni din meniul utilizatorului, toate de tipul *public void*, iar două dintre ele conțin și o aruncare de excepție, mai precis funcția *load_books()* și *add_books()*. Tot în această clasă am și main-ul, în care apelez metodele de adăugare a unei cărți, de sortare, etc.

Cu scopul de a structura biblioteca cât mai bine, m-am ajutat de clasa creată anterior pentru a implementa un vector de cărți, de dimensiunea *MAX_BOOKS*, instanțiată mai sus cu valoarea de 10000. Această variabilă este de tipul *public static final int*, deoarece este o constantă fixă. Static

permite să fie o singură copie a valorii în toată memoria programului. In caz contrar, fiecare obiect de tip “Library” ar rezerva propriul spațiu pentru această valoare. Final este folosit pentru a permite să fie modificată valoarea după ce a fost setată.

```
134 public class Library {
135     public static final int MAX_BOOKS = 10000;
136 
137     private Book[] library = new Book[MAX_BOOKS];
138     private int nr_books = 0;
139 
140     File file = new File("carti.txt");
141 
142     Scanner sc = new Scanner(System.in);
143 
144     private String decor = "";
145 
146     public void load_books() throws IncorrectNrofDetailsException, IOException{
183     }
184 
185     public void save_books() {
200     }
201 
202     public void add_books() throws LibraryFullException {
296     }
297 
298     public void show_details_book(String title) {
323     }
324 
325 
326     public void show_sorted_books(int filter) {
463     }
464 
465     public void edit_book(String title) {
606     }
607 
608     public void delete_books() {
635     }
636 
637 
638     public static void main(String[] args) {
639         Library lib = new Library();
640 
```

Am luat în calcul și varianta de eroare care poate apărea în caz că fișierul nu există.

```
148 
149         if(!file.exists()) {
150             System.out.println("fisierul carti.txt nu exista!!");
151             return;
152         }
153 
```

Datorită faptului că lucrez în principal cu varibile de tip *String*, am utilizat foarte mult metoda oferită de Java, anume *equalsIgnoreCase()*, pentru a evita situații neplacute în care utilizatorul folosește și majuscule și minuscule. Metoda îmi permite să compar două variabile *String*, indiferent de tipul de literă folosită.

```
238             if(status.equalsIgnoreCase("citita") || status.equalsIgnoreCase("wishlist"))
239                 break;
```

Altă metodă ajutătoare din Java este *hasNextInt()*, care verifică dacă următoarele caractere sunt variabile de tip *int*.

```
248                 if(sc.hasNextInt()) {
249                     nr_pages = sc.nextInt();
250                     sc.nextLine();
251
```

Pe deasupra, pentru o interfață cât mai plăcută și lizibilă pentru utilizatori, am introdus o varibilă de tip *String* ce conține caractere decorative și am folsit metoda *repeat()*, cu scopul de a eficientiza codul.

```
373         case 3:
374             System.out.println(decor.repeat(20) + " Carti sortate dupa format " + decor.repeat(20));
375
376             System.out.println(decor.repeat(4) + " Format fizic " + decor.repeat(4));
377             for(int i = 0; i < nr_books; i++) {
378                 if(library[i].getFormat().equalsIgnoreCase("fizic"))
379                     System.out.println("\"" + library[index[i]].getTitle() + "\"" + " || " + library[index[i]].getAuthor());
380             }
```

Cu scopul de a permite o rulare cât mai lină a programului, am introdus bucle *While()*, care permit utilizatorului sî introducă de la tastatură valoarea cerută, până este conform parametrilor setați de mine.

```
245             int nr_pages;
246             while(true) {
247                 System.out.print("Numar de pagini: ");
248                 if(sc.hasNextInt()) {
249                     nr_pages = sc.nextInt();
250                     sc.nextLine();
251
252                 if(nr_pages > 0)
253                     break;
254                 }
255                 System.out.println("Numar invalid! Trebuie sa fie pozitiv.");
256                 System.out.println();
257             }
258             new_book.setNr_pages(nr_pages);
```

M-am folosit de blocul *try...catch* pentru a gestiona excepțiile.

```
641         try {
642             lib.load_books();
643
644             System.out.println("Bun venit la The Book Nook, biblioteca ta digitala!");
645             System.out.println(lib.decor.repeat(30));
646
647             boolean run = true;
648             int option_menu;
649             String y_n;
650
651             while(run) {
770             }
771         }catch(IncorrectNrofDetailsException e) {
772             System.out.println(e.getMessage());
773         }catch(IOException e) {
774             System.out.println("Eroare la citirea fisierului!");
775         }catch(Exception e) {
776             System.out.println("A aparut o eroare neasteptata: " +  e.getMessage());
777         }
```

IV. CONCLUZII

IV.1. CONCLUZII

În primul rând, pe parcursul dezvoltării aplicației “The Book Nook” am reușit să îmi ating toate obiectivele inițial propuse. În final, am creat o aplicație care gestionează eficient și ușor toate cărțile din biblioteca unui utilizator, creând astfel un meniu interactiv cu multe opțiuni pentru acesta.

În al doilea rând, am reușit să creez această aplicație pe cont propriu, ajutându-mă de cunoștințele deja avute, dar și de materiale din cursuri, laboratoare sau de pe site-uri specializate pe limbaje de programare. După gândirea unui plan bine structurat, am parcurs pas cu pas etapele acestuia, iar organizarea bine pusă la punct se reflectă și în codul aplicației, acesta fiind structurat în funcții diverse, care mai apoi doar sunt apelate.

De asemenea, prin intermediul acestui proiect, am avut ocazia să îmi dezvolt abilitățile de programare și cunoștințele, dar și să învăț să folosesc mediul de dezvoltare Eclipse. AM descoperit metode simple și foarte ajutătoare oferite de Java, am învățat să lucrez cu obiecte și clase, POO fiind un concept foarte important în programare și am înțeles mai bine conceptul de excepții și erori și cum să le gestionez.

În concluzie, petrecând mult timp dezvoltând acest program interactiv, mi-am aprofundat cunoștințele în materie de programare, în special limbajul de programare Java.

IV.2. PERSPECTIVE DE VIITOR

Rezultatul obținut până în acest moment este unul satisfăcător, deoarece mi-am atins cu succes toate obiectivele de la început, însă am descoperit pe parcurs faptul că îmi place să dezvolt aplicații, care sunt mai ales legate de interesele mele. Îmi place să mă gândesc cum pot oferi experiențe cât mai plăcute utilizatorilor.

O primă idee pentru viitor este să dezvolt aplicația și mai mult, având poate o bază de date și pentru utilizatori, care se pot loga cu username și parolă, care se pot împrieteni pe platformă și chiar să existe și o secțiune de chat pentru a-și împărtăși gândurile legate de cărțile citite.

Altă idee ar fi să realizez o pagină specială pentru urmărirea obiectivelor pe care utilizatorii și le propun. Dacă, de exemplu, își propune să citească 100 cărți pe an, mi-ar plăcea să aibă parte de grafice interactive și detaliate.

V. BIBLIOGRAFIE

1. https://codecool.com/ro/blog/ghid-java-incepatori/
2. https://wiki.dcae.pub.ro/index.php/Noțiuni_despre_Java
3. https://wiki.dcae.pub.ro/index.php/Tratarea_excepțiilor
4. https://itlectures.ro/wp-content/uploads/2019/10/LABORATOR-1-POO.FAIMA_.v2.pdf
5. https://www.opensourceforu.com/2023/05/eclipse-ide-an-overview/
6. https://www.cogentinfo.com/resources/what-is-eclipse-ide-java-101
7. https://www.w3schools.com
8. Laboratoare și cursuri de la facultate