

JAVA ASSIGNMENT-4

Name-Nandini Kumari

Roll no-2401201085

Course- BCA(AI&DS)

Input-

```
1  import java.io.*;
2  import java.util.*;
3  import java.util.stream.Collectors;
4
5
6  public class LibrarySystem {
7
8
9      static class Book implements Comparable<Book> {
10         private Integer bookId;
11         private String title;
12         private String author;
13         private String category;
14         private boolean isIssued;
15
16         public Book(Integer bookId, String title, String author, String category, boolean isIssued) {
17             this.bookId = bookId;
18             this.title = title;
19             this.author = author;
20             this.category = category;
21             this.isIssued = isIssued;
22         }
23
24         public Integer getBookId() { return bookId; }
25         public String getTitle() { return title; }
26         public String getAuthor() { return author; }
27         public String getCategory() { return category; }
28         public boolean isIssued() { return isIssued; }
```

```

29
30     public void markAsIssued() { isIssued = true; }
31     public void markAsReturned() { isIssued = false; }
32
33     public void displayBookDetails() {
34         System.out.printf(format: "ID: %d | Title: %s | Author: %s | Category: %s | Issued: %s\n",
35             bookid, title, author, category, isIssued ? "Yes" : "No");
36     }
37
38
39     @Override
40     public int compareTo(Book other) {
41         return this.title.compareToIgnoreCase(other.title);
42     }
43
44
45     public String toFileLine() {
46         return String.format(format: "%d|%s|%s|%s|%s",
47             bookid,
48             escapePipe(title),
49             escapePipe(author),
50             escapePipe(category),
51             Boolean.toString(isIssued));
52     }
53
54     public static Book fromFileLine(String line) {

```

```

55         String[] parts = line.split(regex: "\\|", -1);
56         if (parts.length < 5) return null;
57         Integer id = Integer.parseInt(parts[0]);
58         String title = unescapePipe(parts[1]);
59         String author = unescapePipe(parts[2]);
60         String category = unescapePipe(parts[3]);
61         boolean issued = Boolean.parseBoolean(parts[4]);
62         return new Book(id, title, author, category, issued);
63     }
64
65     private static String escapePipe(String s) {
66         return s == null ? "" : s.replace(target: "|", replacement: "\\|");
67     }
68     private static String unescapePipe(String s) {
69         return s == null ? "" : s.replace(target: "\\|", replacement: "|");
70     }
71 }
72
73
74 static class Member {
75     private Integer memberId;
76     private String name;
77     private String email;
78     private List<Integer> issuedBooks;
79

```

```

80     public Member(Integer memberId, String name, String email) {
81         this.memberId = memberId;
82         this.name = name;
83         this.email = email;
84         this.issuedBooks = new ArrayList<>();
85     }
86
87     public Integer getMemberId() { return memberId; }
88     public String getName() { return name; }
89     public String getEmail() { return email; }
90     public List<Integer> getIssuedBooks() { return issuedBooks; }
91
92     public void displayMemberDetails() {
93         System.out.printf(format: "ID: %d | Name: %s | Email: %s | IssuedBooks: %s\n",
94             memberId, name, email,
95             issuedBooks.isEmpty() ? "None" : issuedBooks.toString());
96     }
97
98     public void addIssuedBook(int bookId) {
99         if (!issuedBooks.contains(bookId)) issuedBooks.add(bookId);
100     }
101
102     public void returnIssuedBook(int bookId) {
103         issuedBooks.remove(Integer.valueOf(bookId));
104     }
105

```

```

106     public String toFileLine() {
107         String issued = issuedBooks.stream()
108             .map(Object::toString)
109             .collect(Collectors.joining(delimiter: ","));
110         return String.format(format: "%d|%s|%s|%s",
111             memberId,
112             escapePipe(name),
113             escapePipe(email),
114             issued);
115     }
116
117     public static Member fromFileLine(String line) {
118         String[] parts = line.split(regex: "\\|", -1);
119         if (parts.length < 4) return null;
120         Integer id = Integer.parseInt(parts[0]);
121         String name = unescapePipe(parts[1]);
122         String email = unescapePipe(parts[2]);
123         Member m = new Member(id, name, email);
124         String issued = parts[3];
125         if (!issued.trim().isEmpty()) {
126             String[] ids = issued.split(regex: ",");
127             for (String s : ids) {
128                 try {
129                     m.issuedBooks.add(Integer.parseInt(s.trim()));
130                 } catch (NumberFormatException ignored) {}
131             }
132         }
133     }

```

```

136     private static String escapePipe(String s) {
137         return s == null ? "" : s.replace(target: "|", replacement: "\\|");
138     }
139     private static String unescapePipe(String s) {
140         return s == null ? "" : s.replace(target: "\\|", replacement: "|");
141     }
142 }
143
144
145 static class LibraryManager {
146     private Map<Integer, Book> books = new HashMap<>();
147     private Map<Integer, Member> members = new HashMap<>();
148     private Set<String> categories = new HashSet<>();
149
150     private static final String BOOKS_FILE = "books.txt";
151     private static final String MEMBERS_FILE = "members.txt";
152
153
154     private int nextBookId = 100;
155     private int nextMemberId = 200;
156
157     public LibraryManager() {
158         ensureFilesExist();
159         loadFromFile();
160         recalcNextIds();
161     }

```

```

163     private void ensureFilesExist() {
164         try {
165             new File(BOOKS_FILE).createNewFile();
166             new File(MEMBERS_FILE).createNewFile();
167         } catch (IOException e) {
168             System.err.println("Error ensuring files: " + e.getMessage());
169         }
170     }
171
172     private void recalcNextIds() {
173         if (!books.isEmpty()) nextBookId = Collections.max(books.keySet()) + 1;
174         if (!members.isEmpty()) nextMemberId = Collections.max(members.keySet()) + 1;
175     }
176
177
178     public Book addBook(String title, String author, String category) {
179         Book b = new Book(nextBookId++, title, author, category, isIssued: false);
180         books.put(b.getBookId(), b);
181         categories.add(category);
182         saveBooksToFile();
183         return b;
184     }
185
186     public Member addMember(String name, String email) {
187         Member m = new Member(nextMemberId++, name, email);
188         members.put(m.getMemberId(), m);

```

```

189         saveMembersToFile();
190         return m;
191     }
192
193
194     public boolean issueBook(int bookId, int memberId) {
195         Book b = books.get(bookId);
196         Member m = members.get(memberId);
197         if (b == null) {
198             System.out.println(x: "Book ID not found.");
199             return false;
200         }
201         if (m == null) {
202             System.out.println(x: "Member ID not found.");
203             return false;
204         }
205         if (b.isIssued()) {
206             System.out.println(x: "Book is already issued.");
207             return false;
208         }
209         b.markAsIssued();
210         m.addIssuedBook(bookId);
211         saveBooksToFile();
212         saveMembersToFile();
213         return true;

```

```

214     }
215
216     public boolean returnBook(int bookId, int memberId) {
217         Book b = books.get(bookId);
218         Member m = members.get(memberId);
219         if (b == null) {
220             System.out.println(x: "Book ID not found.");
221             return false;
222         }
223         if (m == null) {
224             System.out.println(x: "Member ID not found.");
225             return false;
226         }
227         if (!b.isIssued()) {
228             System.out.println(x: "Book is not marked as issued.");
229             return false;
230         }
231         if (!m.getIssuedBooks().contains(bookId)) {
232             System.out.println(x: "This member does not have that book issued.");
233             return false;
234         }
235         b.markAsReturned();
236         m.returnIssuedBook(bookId);
237         saveBooksToFile();
238         saveMembersToFile();

```

```

239         return true;
240     }
241
242
243     public List<Book> searchBooks(String keyword, String mode) {
244         String k = keyword.toLowerCase();
245         List<Book> results = new ArrayList<>();
246         for (Book b : books.values()) {
247             switch (mode.toLowerCase()) {
248                 case "title":
249                     if (b.getTitle().toLowerCase().contains(k)) results.add(b);
250                     break;
251                 case "author":
252                     if (b.getAuthor().toLowerCase().contains(k)) results.add(b);
253                     break;
254                 case "category":
255                     if (b.getCategory().toLowerCase().contains(k)) results.add(b);
256                     break;
257                 default:
258
259                     if (b.getTitle().toLowerCase().contains(k) ||
260                         b.getAuthor().toLowerCase().contains(k) ||
261                         b.getCategory().toLowerCase().contains(k)) results.add(b);
262             }
263         }

```

```

264         return results;
265     }
266
267
268     public List<Book> sortBooksByTitle(boolean ascending) {
269         List<Book> list = new ArrayList<>(books.values());
270         list.sort(ascending ? Comparator.naturalOrder() : Comparator.reverseOrder());
271         return list;
272     }
273
274
275     public List<Book> sortBooksByAuthor(boolean ascending) {
276         List<Book> list = new ArrayList<>(books.values());
277         list.sort((a, b) -> {
278             int cmp = a.getAuthor().compareToIgnoreCase(b.getAuthor());
279             return ascending ? cmp : -cmp;
280         });
281         return list;
282     }
283
284
285     public List<Book> sortBooksByCategory(boolean ascending) {
286         List<Book> list = new ArrayList<>(books.values());
287         list.sort((a, b) -> {
288             int cmp = a.getCategory().compareToIgnoreCase(b.getCategory());

```

```

289         return ascending ? cmp : -cmp;
290     });
291     return list;
292 }
293
294
295 public void loadFromFile() {
296     loadBooksFromFile();
297     loadMembersFromFile();
298 }
299
300 private void loadBooksFromFile() {
301     try (BufferedReader br = new BufferedReader(new FileReader(BOOKS_FILE))) {
302         String line;
303         while ((line = br.readLine()) != null) {
304             line = line.trim();
305             if (line.isEmpty()) continue;
306             Book b = Book.fromFileLine(line);
307             if (b != null) {
308                 books.put(b.getBookId(), b);
309                 categories.add(b.getCategory());
310             }
311         }
312     } catch (IOException e) {
313         System.err.println("Error loading books: " + e.getMessage());

```

```

314     }
315 }
316
317 private void loadMembersFromFile() {
318     try (BufferedReader br = new BufferedReader(new FileReader(MEMBERS_FILE))) {
319         String line;
320         while ((line = br.readLine()) != null) {
321             line = line.trim();
322             if (line.isEmpty()) continue;
323             Member m = Member.fromFileLine(line);
324             if (m != null) members.put(m.getMemberId(), m);
325         }
326     } catch (IOException e) {
327         System.err.println("Error loading members: " + e.getMessage());
328     }
329 }
330
331 private void saveBooksToFile() {
332     try (BufferedWriter bw = new BufferedWriter(new FileWriter(BOOKS_FILE, append: false))) {
333         for (Book b : books.values()) {
334             bw.write(b.toFileLine());
335             bw.newLine();
336         }
337     } catch (IOException e) {
338         System.err.println("Error saving books: " + e.getMessage());

```

```

339     }
340 }
341
342 private void saveMembersToFile() {
343     try (BufferedWriter bw = new BufferedWriter(new FileWriter(MEMBERS_FILE, append: false))) {
344         for (Member m : members.values()) {
345             bw.write(m.toFileLine());
346             bw.newLine();
347         }
348     } catch (IOException e) {
349         System.err.println("Error saving members: " + e.getMessage());
350     }
351 }
352
353 public void saveAll() {
354     saveBooksToFile();
355     saveMembersToFile();
356 }
357
358
359 public Optional<Book> getBookById(int id) {
360     return Optional.ofNullable(books.get(id));
361 }
362 public Optional<Member> getMemberById(int id) {
363     return Optional.ofNullable(members.get(id));
364 }

```

```

364     }
365     public Set<String> getCategories() { return categories; }
366     public Collection<Book> getAllBooks() { return books.values(); }
367     public Collection<Member> getAllMembers() { return members.values(); }
368 }
369
370

```

```

Run | Debug
371 public static void main(String[] args) {
372     Scanner sc = new Scanner(System.in);
373     LibraryManager lm = new LibraryManager();
374
375     System.out.println(x: "Welcome to City Library Digital Management System");
376
377     boolean exit = false;
378     while (!exit) {
379         System.out.println(x: "\n1. Add Book");
380         System.out.println(x: "2. Add Member");
381         System.out.println(x: "3. Issue Book");
382         System.out.println(x: "4. Return Book");
383         System.out.println(x: "5. Search Books");
384         System.out.println(x: "6. Sort Books");
385         System.out.println(x: "7. List All Books");
386         System.out.println(x: "8. List All Members");
387         System.out.println(x: "9. Exit");

```

```

388     System.out.print(s: "Enter your choice: ");
389     String choice = sc.nextLine().trim();
390
391     switch (choice) {
392     case "1":
393         System.out.print(s: "Enter Book Title: ");
394         String title = sc.nextLine().trim();
395         System.out.print(s: "Enter Author: ");
396         String author = sc.nextLine().trim();
397         System.out.print(s: "Enter Category: ");
398         String category = sc.nextLine().trim();
399         Book b = lm.addBook(title, author, category);
400         System.out.println("Book added successfully with ID: " + b.getBookId());
401         break;
402
403     case "2":
404         System.out.print(s: "Enter Member Name: ");
405         String name = sc.nextLine().trim();
406         System.out.print(s: "Enter Email: ");
407         String email = sc.nextLine().trim();
408         Member m = lm.addMember(name, email);
409         System.out.println("Member added successfully with ID: " + m.getMemberId());
410         break;
411
412     case "3":
413         System.out.print(s: "Enter Book ID to issue: ");

```

```

414         int bidIssue = parseIntInput(sc.nextLine());
415         System.out.print(s: "Enter Member ID: ");
416         int midIssue = parseIntInput(sc.nextLine());
417         if (lm.issueBook(bidIssue, midIssue)) {
418             System.out.println(x: "Book issued successfully.");
419         } else {
420             System.out.println(x: "Issue failed.");
421         }
422         break;
423
424     case "4":
425         System.out.print(s: "Enter Book ID to return: ");
426         int bidReturn = parseIntInput(sc.nextLine());
427         System.out.print(s: "Enter Member ID: ");
428         int midReturn = parseIntInput(sc.nextLine());
429         if (lm.returnBook(bidReturn, midReturn)) {
430             System.out.println(x: "Book returned successfully.");
431         } else {
432             System.out.println(x: "Return failed.");
433         }
434         break;
435
436     case "5":
437         System.out.print(s: "Search by (title/author/category/all): ");
438         String mode = sc.nextLine().trim().toLowerCase();
439         System.out.print(s: "Enter keyword: ");

```

```

440         String kw = sc.nextLine().trim();
441         List<Book> results = lm.searchBooks(kw, mode);
442         System.out.println("Search results (" + results.size() + "):");
443         for (Book rb : results) rb.displayBookDetails();
444         break;
445
446     case "6":
447         System.out.println(x: "Sort options: 1-Title 2-Author 3-Category");
448         System.out.print(s: "Choose: ");
449         String sopt = sc.nextLine().trim();
450         System.out.print(s: "Ascending? (y/n): ");
451         boolean asc = sc.nextLine().trim().equalsIgnoreCase(anotherString: "y");
452         List<Book> sorted = new ArrayList<>();
453         if ("1".equals(sopt)) sorted = lm.sortBooksByTitle(asc);
454         else if ("2".equals(sopt)) sorted = lm.sortBooksByAuthor(asc);
455         else if ("3".equals(sopt)) sorted = lm.sortBooksByCategory(asc);
456         else {
457             System.out.println(x: "Invalid option.");
458             break;
459         }
460         System.out.println(x: "Sorted list:");
461         for (Book sb : sorted) sb.displayBookDetails();
462         break;
463
464     case "7":
465         System.out.println(x: "All books:");

```

```

466         for (Book ab : lm.getAllBooks()) ab.displayBookDetails();
467         break;
468
469     case "8":
470         System.out.println(x: "All members:");
471         for (Member mem : lm.getAllMembers()) mem.displayMemberDetails();
472         break;
473
474     case "9":
475         lm.saveAll();
476         System.out.println(x: "Saved data. Exiting...");
477         exit = true;
478         break;
479
480     default:
481         System.out.println(x: "Invalid choice. Try again.");
482     }
483 }
484
485 sc.close();
486 }
487
488 private static int parseIntInput(String s) {
489     try {
490         return Integer.parseInt(s.trim());
491     } catch (NumberFormatException e) {

```

```
482         }
483     }
484
485     sc.close();
486 }
487
488 private static int parseIntInput(String s) {
489     try {
490         return Integer.parseInt(s.trim());
491     } catch (NumberFormatException e) {
492         return -1;
493     }
494 }
495 }
496
```

Output-

```
Welcome to City Library Digital Management System

1. Add Book
2. Add Member
3. Issue Book
4. Return Book
5. Search Books
6. Sort Books
7. List All Books
8. List All Members
9. Exit
```

```
Enter your choice: 1
Enter Book Title: Life
Enter Author: Nandini Kumari
Enter Category: Biography
Book added successfully with ID: 101
```

1. Add Book
2. Add Member
3. Issue Book
4. Return Book
5. Search Books
6. Sort Books

```
7. List All Books
8. List All Members
9. Exit
Enter your choice: 2
Enter Member Name: Nandini Kumari
Enter Email: Nandiniii428@gmail.com
Member added successfully with ID: 201
```

1. Add Book
2. Add Member
3. Issue Book
4. Return Book

```
5. Search Books
6. Sort Books
7. List All Books
8. List All Members
9. Exit
Enter your choice: 3
Enter Book ID to issue: 101
Enter Member ID: 201
Book issued successfully.
```

1. Add Book
2. Add Member

```
3. Issue Book
4. Return Book
5. Search Books
6. Sort Books
7. List All Books
8. List All Members
9. Exit
Enter your choice: 4
Enter Book ID to return: 101
Enter Member ID: 201
Book returned successfully.
```

```
1. Add Book
2. Add Member
3. Issue Book
4. Return Book
5. Search Books
6. Sort Books
7. List All Books
8. List All Members
9. Exit
Enter your choice: 5
Search by (title/author/category/all): author
```

```
Enter keyword: Nandini Kumari
Search results (1):
ID: 101 | Title: Life | Author: Nandini Kumari | Category: Biography | Issued: No

1. Add Book
2. Add Member
3. Issue Book
4. Return Book
5. Search Books
6. Sort Books
7. List All Books
8. List All Members
```

8. List All Members

9. Exit

Enter your choice: 6

Sort options: 1-Title 2-Author 3-Category

Choose: Title

Ascending? (y/n): n

Invalid option.

1. Add Book

2. Add Member

3. Issue Book

4. Return Book

4. Return Book

5. Search Books

6. Sort Books

7. List All Books

8. List All Members

9. Exit

Enter your choice: 7

All books:

4. Return Book

5. Search Books

6. Sort Books

7. List All Books

8. List All Members

9. Exit

Enter your choice: 7

All books:

ID: 201 | Name: Nandini Kumari | Email: Nandiniii428@gmail.com | IssuedBooks: None

1. Add Book

2. Add Member

3. Issue Book

4. Return Book

5. Search Books

6. Sort Books

7. List All Books

```
6. Sort Books
7. List All Books
8. List All Members
9. Exit
Enter your choice: 9
Saved data. Exiting...
```

EXPLANATION

1. Book Class (Represents each book)

This part of the program creates a “Book object” with details like:

- *ID*
- *Title*
- *Author*
- *Category*
- *Whether the book is issued or not*

It also has functions to:

- *Show book details*
- *Mark a book as issued*
- *Mark a book as returned*

The book can also save its details to a file and read them back.

2. Member Class (Represents a library member)

Every member has:

- *ID*
- *Name*

- *Email*
- *A list of books they have taken*

It has functions to:

- *Display member details*
- *Add a book to their issued list*
- *Remove a returned book from their list*

This class also knows how to save and load its data from the file.

3. *LibraryManager Class (Main controller of the whole program)*

This is the brain of the system.

It manages:

- *A collection of all books (using a Map)*
- *A collection of all members (using another Map)*
- *A Set of categories*

It performs major tasks like:

- *Adding books and members*
- *Issuing or returning books*
- *Searching books*
- *Sorting books by title, author, or category*
- *Loading data from the text files when the program starts*
- *Saving data back to the text files when anything changes*

It uses `BufferedReader/Writer` for fast reading/writing.

4. *Main() Method (The menu the user sees)*

This is the part that runs when you start the program.

It shows a menu like:

1. Add Book

2. Add Member

3. Issue Book

4. Return Book

5. Search Books

6. Sort Books

7. Exit

You type a number → the program runs that operation.

This menu keeps showing until you choose Exit.