JAVA ASSIGNMENT-4

Name-Rajat Rao

Roll no.-2401201067

Course- BCA(AI&DS)

Input-

```java
import java.io.*;
import java.util.*;
import java.util.stream.Collectors;


public class LibrarySystem {


    static class Book implements Comparable<Book> {
        private Integer bookId;
        private String title;
        private String author;
        private String category;
        private boolean isIssued;

        public Book(Integer bookId, String title, String author, String category, boolean isIssued) {
            this.bookId = bookId;
            this.title = title;
            this.author = author;
            this.category = category;
            this.isIssued = isIssued;
        }

        public Integer getBookId() { return bookId; }
        public String getTitle() { return title; }
        public String getAuthor() { return author; }
        public String getCategory() { return category; }
        public boolean isIssued() { return isIssued; }
```

```java

    public void markAsIssued() { isIssued = true; }
    public void markAsReturned() { isIssued = false; }

    public void displayBookDetails() {
        System.out.printf(format: "ID: %d | Title: %s | Author: %s | Category: %s | Issued: %s%n",
                bookId, title, author, category, isIssued ? "Yes" : "No");
    }


    @Override
    public int compareTo(Book other) {
        return this.title.compareToIgnoreCase(other.title);
    }


    public String toFileLine() {
        return String.format(format: "%d|%s|%s|%s|%s",
                bookId,
                escapePipe(title),
                escapePipe(author),
                escapePipe(category),
                Boolean.toString(isIssued));
    }

    public static Book fromFileLine(String line) {
```


```java

    public void markAsIssued() { isIssued = true; }
    public void markAsReturned() { isIssued = false; }

    public void displayBookDetails() {
        System.out.printf(format: "ID: %d | Title: %s | Author: %s | Category: %s | Issued: %s%n",
                bookId, title, author, category, isIssued ? "Yes" : "No");
    }


    @Override
    public int compareTo(Book other) {
        return this.title.compareToIgnoreCase(other.title);
    }


    public String toFileLine() {
        return String.format(format: "%d|%s|%s|%s|%s",
                bookId,
                escapePipe(title),
                escapePipe(author),
                escapePipe(category),
                Boolean.toString(isIssued));
    }

    public static Book fromFileLine(String line) {
```

```java
    public Member(Integer memberId, String name, String email) {
        this.memberId = memberId;
        this.name = name;
        this.email = email;
        this.issuedBooks = new ArrayList<>();
    }

    public Integer getMemberId() { return memberId; }
    public String getName() { return name; }
    public String getEmail() { return email; }
    public List<Integer> getIssuedBooks() { return issuedBooks; }

    public void displayMemberDetails() {
        System.out.printf(format: "ID: %d | Name: %s | Email: %s | IssuedBooks: %s%n",
                memberId, name, email,
                issuedBooks.isEmpty() ? "None" : issuedBooks.toString());
    }

    public void addIssuedBook(int bookId) {
        if (!issuedBooks.contains(bookId)) issuedBooks.add(bookId);
    }

    public void returnIssuedBook(int bookId) {
        issuedBooks.remove(Integer.valueOf(bookId));
    }

```

```java

    public String toFileLine() {
        String issued = issuedBooks.stream()
                .map(Object::toString)
                .collect(Collectors.joining(delimiter: ","));
        return String.format(format: "%d|%s|%s|%s",
                memberId,
                escapePipe(name),
                escapePipe(email),
                issued);
    }

    public static Member fromFileLine(String line) {
        String[] parts = line.split(regex: "\\|", -1);
        if (parts.length < 4) return null;
        Integer id = Integer.parseInt(parts[0]);
        String name = unescapePipe(parts[1]);
        String email = unescapePipe(parts[2]);
        Member m = new Member(id, name, email);
        String issued = parts[3];
        if (!issued.trim().isEmpty()) {
            String[] ids = issued.split(regex: ",");
            for (String s : ids) {
                try {
                    m.issuedBooks.add(Integer.parseInt(s.trim()));
                } catch (NumberFormatException ignored) {}
```

```
            }
        }
        return m;
    }

    private static String escapePipe(String s) {
        return s == null ? "" : s.replace(target: "|", replacement: "\\|");
    }
    private static String unescapePipe(String s) {
        return s == null ? "" : s.replace(target: "\\|", replacement: "|");
    }
}


static class LibraryManager {
    private Map<Integer, Book> books = new HashMap<>();
    private Map<Integer, Member> members = new HashMap<>();
    private Set<String> categories = new HashSet<>();

    private static final String BOOKS_FILE = "books.txt";
    private static final String MEMBERS_FILE = "members.txt";


    private int nextBookId = 100;
    private int nextMemberId = 200;
```

```
    public LibraryManager() {
        ensureFilesExist();
        loadFromFile();
        recalcNextIds();
    }

    private void ensureFilesExist() {
        try {
            new File(BOOKS_FILE).createNewFile();
            new File(MEMBERS_FILE).createNewFile();
        } catch (IOException e) {
            System.err.println("Error ensuring files: " + e.getMessage());
        }
    }

    private void recalcNextIds() {
        if (!books.isEmpty()) nextBookId = Collections.max(books.keySet()) + 1;
        if (!members.isEmpty()) nextMemberId = Collections.max(members.keySet()) + 1;
    }


    public Book addBook(String title, String author, String category) {
        Book b = new Book(nextBookId++, title, author, category, isIssued: false);
        books.put(b.getBookId(), b);
        categories.add(category);
        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.");
```

```
            return false;
        }
        b.markAsReturned();
        m.returnIssuedBook(bookId);
        saveBooksToFile();
        saveMembersToFile();
        return true;
    }


    public List<Book> searchBooks(String keyword, String mode) {
        String k = keyword.toLowerCase();
        List<Book> results = new ArrayList<>();
        for (Book b : books.values()) {
            switch (mode.toLowerCase()) {
                case "title":
                    if (b.getTitle().toLowerCase().contains(k)) results.add(b);
                    break;
                case "author":
                    if (b.getAuthor().toLowerCase().contains(k)) results.add(b);
                    break;
                case "category":
                    if (b.getCategory().toLowerCase().contains(k)) results.add(b);
                    break;
                default:
```

```

                    if (b.getTitle().toLowerCase().contains(k) ||
                            b.getAuthor().toLowerCase().contains(k) ||
                            b.getCategory().toLowerCase().contains(k)) results.add(b);
            }
        }
        return results;
    }


    public List<Book> sortBooksByTitle(boolean ascending) {
        List<Book> list = new ArrayList<>(books.values());
        list.sort(ascending ? Comparator.naturalOrder() : Comparator.reverseOrder());
        return list;
    }


    public List<Book> sortBooksByAuthor(boolean ascending) {
        List<Book> list = new ArrayList<>(books.values());
        list.sort((a, b) -> {
            int cmp = a.getAuthor().compareToIgnoreCase(b.getAuthor());
            return ascending ? cmp : -cmp;
        });
        return list;
    }
```

```java

    public List<Book> sortBooksByCategory(boolean ascending) {
        List<Book> list = new ArrayList<>(books.values());
        list.sort((a, b) -> {
            int cmp = a.getCategory().compareToIgnoreCase(b.getCategory());
            return ascending ? cmp : -cmp;
        });
        return list;
    }


    public void loadFromFile() {
        loadBooksFromFile();
        loadMembersFromFile();
    }

    private void loadBooksFromFile() {
        try (BufferedReader br = new BufferedReader(new FileReader(BOOKS_FILE))) {
            String line;
            while ((line = br.readLine()) != null) {
                line = line.trim();
                if (line.isEmpty()) continue;
                Book b = Book.fromFileLine(line);
                if (b != null) {
                    books.put(b.getBookId(), b);
                    categories.add(b.getCategory());
```

```java
                }
            }
        } catch (IOException e) {
            System.err.println("Error loading books: " + e.getMessage());
        }
    }

    private void loadMembersFromFile() {
        try (BufferedReader br = new BufferedReader(new FileReader(MEMBERS_FILE))) {
            String line;
            while ((line = br.readLine()) != null) {
                line = line.trim();
                if (line.isEmpty()) continue;
                Member m = Member.fromFileLine(line);
                if (m != null) members.put(m.getMemberId(), m);
            }
        } catch (IOException e) {
            System.err.println("Error loading members: " + e.getMessage());
        }
    }

    private void saveBooksToFile() {
        try (BufferedWriter bw = new BufferedWriter(new FileWriter(BOOKS_FILE, append: false))) {
            for (Book b : books.values()) {
                bw.write(b.toFileLine());
```

```
                bw.newLine();
            }
        } catch (IOException e) {
            System.err.println("Error saving books: " + e.getMessage());
        }
    }

    private void saveMembersToFile() {
        try (BufferedWriter bw = new BufferedWriter(new FileWriter(MEMBERS_FILE, append: false))) {
            for (Member m : members.values()) {
                bw.write(m.toFileLine());
                bw.newLine();
            }
        } catch (IOException e) {
            System.err.println("Error saving members: " + e.getMessage());
        }
    }

    public void saveAll() {
        saveBooksToFile();
        saveMembersToFile();
    }


    public Optional<Book> getBookById(int id) {
```

```
    public Optional<Book> getBookById(int id) {
        return Optional.ofNullable(books.get(id));
    }
    public Optional<Member> getMemberById(int id) {
        return Optional.ofNullable(members.get(id));
    }
    public Set<String> getCategories() { return categories; }
    public Collection<Book> getAllBooks() { return books.values(); }
    public Collection<Member> getAllMembers() { return members.values(); }
}


Run | Debug
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    LibraryManager lm = new LibraryManager();

    System.out.println(x: "Welcome to City Library Digital Management System");

    boolean exit = false;
    while (!exit) {
        System.out.println(x: "\n1. Add Book");
        System.out.println(x: "2. Add Member");
        System.out.println(x: "3. Issue Book");
        System.out.println(x: "4. Return Book");
        System.out.println(x: "5. Search Books");
        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":
```

```
                System.out.print(s: "Search by (title/author/category/all): ");
                String mode = sc.nextLine().trim().toLowerCase();
                System.out.print(s: "Enter keyword: ");
                String kw = sc.nextLine().trim();
                List<Book> results = lm.searchBooks(kw, mode);
                System.out.println("Search results (" + results.size() + "):");
                for (Book rb : results) rb.displayBookDetails();
                break;

            case "6":
                System.out.println(x: "Sort options: 1-Title 2-Author 3-Category");
                System.out.print(s: "Choose: ");
                String sopt = sc.nextLine().trim();
                System.out.print(s: "Ascending? (y/n): ");
                boolean asc = sc.nextLine().trim().equalsIgnoreCase(anotherString: "y");
                List<Book> sorted = new ArrayList<>();
                if ("1".equals(sopt)) sorted = lm.sortBooksByTitle(asc);
                else if ("2".equals(sopt)) sorted = lm.sortBooksByAuthor(asc);
                else if ("3".equals(sopt)) sorted = lm.sortBooksByCategory(asc);
                else {
                    System.out.println(x: "Invalid option.");
                    break;
                }
                System.out.println(x: "Sorted list:");
                for (Book sb : sorted) sb.displayBookDetails();
                break;
```

```
            case "7":
                System.out.println(x: "All books:");
                for (Book ab : lm.getAllBooks()) ab.displayBookDetails();
                break;

            case "8":
                System.out.println(x: "All members:");
                for (Member mem : lm.getAllMembers()) mem.displayMemberDetails();
                break;

            case "9":
                lm.saveAll();
                System.out.println(x: "Saved data. Exiting...");
                exit = true;
                break;

            default:
                System.out.println(x: "Invalid choice. Try again.");
        }
    }

    sc.close();
}

private static int parseIntInput(String s) {
    try {
```

```
490                 return Integer.parseInt(s.trim());
491             } catch (NumberFormatException e) {
492                 return -1;
493             }
494         }
495     }
496
```

Output-

```
PS C:\Users\rajat> cd "c:\Users\rajat\OneDrive\Desktop\rajat java\" ; if ($?) { javac LibrarySystem.java } ; if ($?) { java LibrarySystem }
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: Rajat Rao
Enter Author: rajat
Enter Category: sci fi
Book added successfully with ID: 100

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: Rajat
Enter Email: rajatrao404@gmail.com
Member added successfully with ID: 200

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: 3
Enter Book ID to issue: 100
Enter Member ID: 200
Book issued successfully.

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

```
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: 100
Enter Member ID: 200
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): title
```

```
Enter keyword: Rajat Rao
Search results (1):
ID: 100 | Title: Rajat Rao | Author: rajat | Category: sci fi | 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
```

```
9. Exit
Enter your choice: 6
Sort options: 1-Title 2-Author 3-Category
Choose: Author
Ascending? (y/n): y
Invalid option.

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: 7
All books:
ID: 100 | Title: Rajat Rao | Author: rajat | Category: sci fi | 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
9. Exit
Enter your choice: 8
All members:
ID: 200 | Name: Rajat | Email: rajatrao404@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
8. List All Members
9. Exit
Enter your choice: 9
Saved data. Exiting...
PS C:\Users\rajat\OneDrive\Desktop\rajat java>
```

EXPLANATION

1. Book Class

- **Represents a single book in the library.**
- **Stores:**
    - **bookId, title, author, category, isIssued**

- **Methods:**
    - **displayBookDetails() → shows book info.**
    - **markAsIssued() / markAsReturned() → update issue status.**
- **Implements Comparable → allows sorting by title.**
- **Can convert itself to/from a single line in books.txt.**

---

2. Member Class

- **Represents a library member.**
- **Stores:**
    - **memberId, name, email, issuedBooks (list of book IDs)**
- **Methods:**
    - **displayMemberDetails() → shows member info.**
    - **addIssuedBook() / returnIssuedBook() → manage issued books.**
- **Also can convert itself to/from a line in members.txt.**

---

3. LibraryManager Class

Main controller that handles all operations.

Stores data using:

- **Map<Integer, Book> books**
- **Map<Integer, Member> members**
- **Set<String> categories**

Features:

- **addBook() / addMember()**
  Creates new objects, stores them in maps & writes to file.
- **issueBook() / returnBook()**
  Updates both book and member data, and saves changes.
- **searchBooks()**
  Searches by title, author, or category.

- **sortBooks()**
  Uses:
  - **Comparable → sort by title**
  - **Comparator → sort by author or category**
- **loadFromFile() / saveToFile()**
  Uses BufferedReader/BufferedWriter to read/write:
  - **books.txt**
  - **members.txt**

---

4. Main Menu

- **A loop that repeatedly shows options:**
- **1. Add Book**
- **2. Add Member**
- **3. Issue Book**
- **4. Return Book**
- **5. Search Books**
- **6. Sort Books**
- **7. Exit**
- **Based on user input, calls respective LibraryManager methods.**