JAVA ASSIGNMENT 4

NAME-BHAVYA RATTAN

ROLL NO-2401201004

COURSE-BCA(AI&DS)

Code:

```java
CityLibraryDigitalManagementSystem.java > ...
public class CityLibraryDigitalManagementSystem {
    static class Book implements Comparable<Book> {
        }

        public static Book fromDataString(String line) {
            String[] parts = line.split(regex: "\\|", -1);
            if (parts.length < 5) return null;
            Integer id = Integer.parseInt(parts[0]);
            String t = parts[1];
            String a = parts[2];
            String c = parts[3];
            boolean issued = Boolean.parseBoolean(parts[4]);
            return new Book(id, t, a, c, issued);
        }
    }

    // ------------------------- MEMBER CLASS -------------------------
    static class Member {
        private Integer memberId;
        private String name;
        private String email;
        private List<Integer> issuedBooks;

        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.println("ID: " + memberId + " | Name: " + name + " | Email: " + email +
                    " | Issued Books: " + issuedBooks);
        }

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

```
        }

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

        public String toDataString() {
            String issued = issuedBooks.stream()
                    .map(String::valueOf)
                    .collect(Collectors.joining(delimiter: ","));
            return memberId + "|" + name + "|" + email + "|" + issued;
        }

        public static Member fromDataString(String line) {
            String[] parts = line.split(regex: "\\|", -1);
            if (parts.length < 4) return null;
            Integer id = Integer.parseInt(parts[0]);
            String n = parts[1];
            String e = parts[2];
            String issued = parts[3];
            List<Integer> issuedList = new ArrayList<>();
            if (!issued.isEmpty()) {
                for (String s : issued.split(regex: ",")) {
                    if (!s.trim().isEmpty()) issuedList.add(Integer.parseInt(s.trim()));
                }
            }
            Member m = new Member(id, n, e);
            m.issuedBooks = issuedList;
            return m;
        }
    }

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

        private final String booksFile = "books.txt";
        private final String membersFile = "members.txt";
        private int nextBookId = 100;
        private int nextMemberId = 200;
```

```java

    public LibraryManager() {
        loadFromFile();
    }

    public Book addBook(String title, String author, String category) {
        int id = nextBookId++;
        Book b = new Book(id, title, author, category, isIssued: false);
        books.put(id, b);
        categories.add(category);
        saveBooks();
        return b;
    }

    public Member addMember(String name, String email) {
        int id = nextMemberId++;
        Member m = new Member(id, name, email);
        members.put(id, m);
        saveMembers();
        return m;
    }

    public boolean issueBook(int bookId, int memberId) {
        Book b = books.get(bookId);
        Member m = members.get(memberId);
        if (b == null || m == null || b.isIssued()) return false;
        b.markAsIssued();
        m.addIssuedBook(bookId);
        saveAll();
        return true;
    }

    public boolean returnBook(int bookId, int memberId) {
        Book b = books.get(bookId);
        Member m = members.get(memberId);
        if (b == null || m == null || !b.isIssued()) return false;
        b.markAsReturned();
        m.returnIssuedBook(bookId);
        saveAll();
        return true;
    }

```

```
static class LibraryManager {
    public List<Book> searchBooks(String keyword, String by) {
        String k = keyword.toLowerCase();
        List<Book> result = new ArrayList<>();
        for (Book b : books.values()) {
            switch (by.toLowerCase()) {
                case "title":
                    if (b.getTitle().toLowerCase().contains(k)) result.add(b);
                    break;
                case "author":
                    if (b.getAuthor().toLowerCase().contains(k)) result.add(b);
                    break;
                case "category":
                    if (b.getCategory().toLowerCase().contains(k)) result.add(b);
                    break;
                default:
                    if (b.getTitle().toLowerCase().contains(k) ||
                            b.getAuthor().toLowerCase().contains(k) ||
                            b.getCategory().toLowerCase().contains(k))
                        result.add(b);
            }
        }
        return result;
    }

    public List<Book> sortBooks(String by) {
        List<Book> list = new ArrayList<>(books.values());
        switch (by.toLowerCase()) {
            case "author":
                list.sort(Comparator.comparing(Book::getAuthor, String.CASE_INSENSITIVE_ORDER));
                break;
            case "category":
                list.sort(Comparator.comparing(Book::getCategory, String.CASE_INSENSITIVE_ORDER));
                break;
            default:
                Collections.sort(list);
        }
        return list;
    }

    public void loadFromFile() {
        loadBooks();
        loadMembers();
```

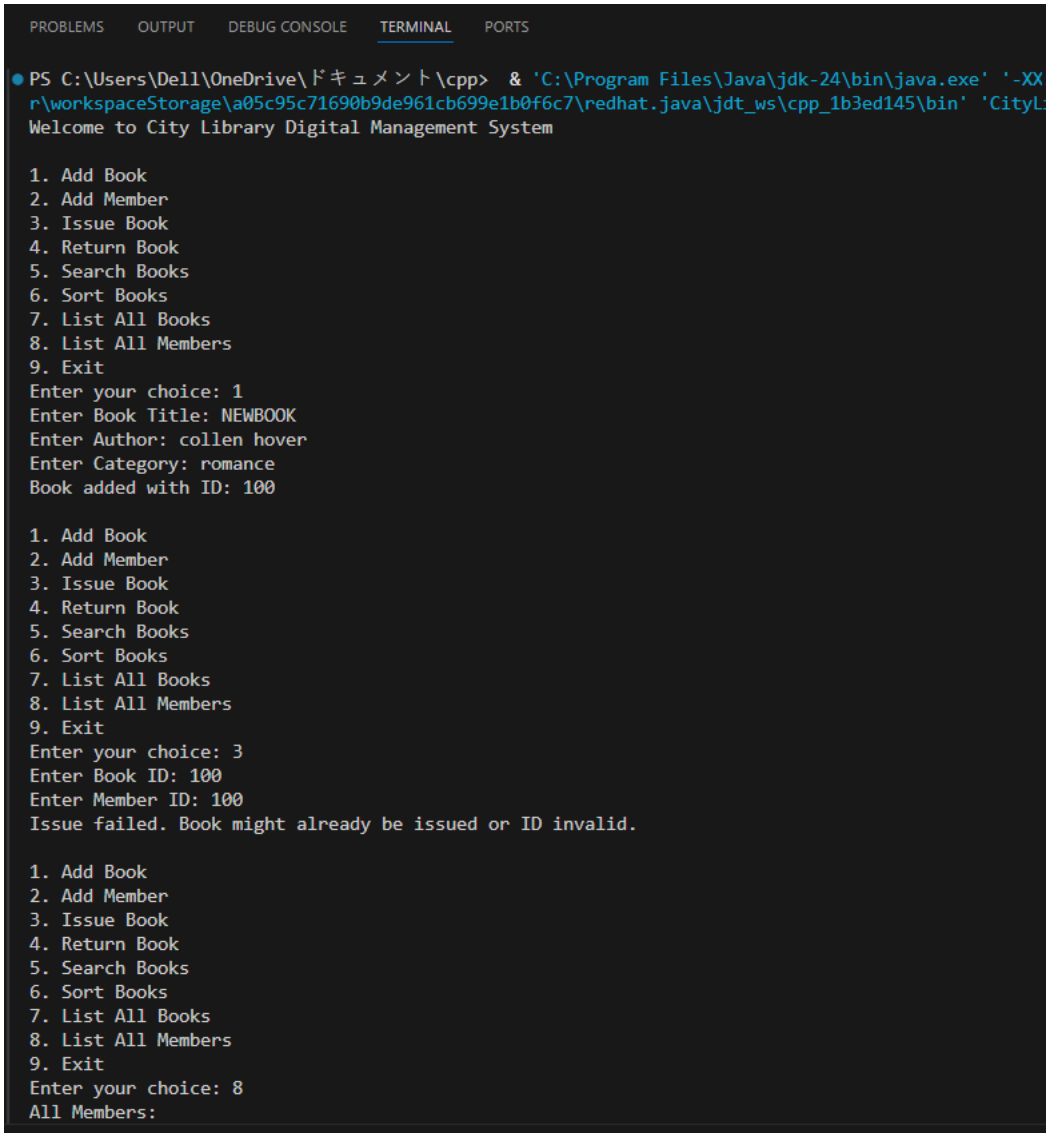
```
118     static class LibraryManager {
208         public void loadFromFile() {
213         }
214
215         private void loadBooks() {
216             books.clear();
217             Path path = Paths.get(booksFile);
218             if (!Files.exists(path)) createFile(path);
219             try (BufferedReader br = Files.newBufferedReader(path)) {
220                 String line;
221                 while ((line = br.readLine()) != null) {
222                     if (line.trim().isEmpty()) continue;
223                     Book b = Book.fromDataString(line);
224                     if (b != null) {
225                         books.put(b.getBookId(), b);
226                         categories.add(b.getCategory());
227                     }
228                 }
229             } catch (IOException e) { System.err.println("Error loading books: " + e.getMessage()); }
230         }
231
232         private void loadMembers() {
233             members.clear();
234             Path path = Paths.get(membersFile);
235             if (!Files.exists(path)) createFile(path);
236             try (BufferedReader br = Files.newBufferedReader(path)) {
237                 String line;
238                 while ((line = br.readLine()) != null) {
239                     if (line.trim().isEmpty()) continue;
240                     Member m = Member.fromDataString(line);
241                     if (m != null) members.put(m.getMemberId(), m);
242                 }
243             } catch (IOException e) { System.err.println("Error loading members: " + e.getMessage()); }
244         }
245
246         private void saveBooks() {
247             try (BufferedWriter bw = Files.newBufferedWriter(Paths.get(booksFile))) {
248                 for (Book b : books.values()) {
249                     bw.write(b.toDataString());
250                     bw.newLine();
251                 }
252             } catch (IOException e) { System.err.println("Error saving books: " + e.getMessage()); }
253         }
```

```
118     static class LibraryManager {
254
255         private void saveMembers() {
256             try (BufferedWriter bw = Files.newBufferedWriter(Paths.get(membersFile))) {
257                 for (Member m : members.values()) {
258                     bw.write(m.toDataString());
259                     bw.newLine();
260                 }
261             } catch (IOException e) { System.err.println("Error saving members: " + e.getMessage()); }
262         }
263
264         public void saveAll() {
265             saveBooks();
266             saveMembers();
267         }
268
269         private void createFile(Path path) {
270             try { Files.createFile(path); }
271             catch (IOException e) { System.err.println("Cannot create file: " + e.getMessage()); }
272         }
273
274         public Collection<Book> getAllBooks() { return books.values(); }
275         public Collection<Member> getAllMembers() { return members.values(); }
276     }
277
278     // ------------------------- MAIN MENU -------------------------
        Run | Debug
279     public static void main(String[] args) {
280         Scanner sc = new Scanner(System.in);
281         LibraryManager manager = new LibraryManager();
282         System.out.println(x: "Welcome to City Library Digital Management System");
283
284         while (true) {
285             System.out.println(x: "\n1. Add Book");
286             System.out.println(x: "2. Add Member");
287             System.out.println(x: "3. Issue Book");
288             System.out.println(x: "4. Return Book");
289             System.out.println(x: "5. Search Books");
290             System.out.println(x: "6. Sort Books");
291             System.out.println(x: "7. List All Books");
292             System.out.println(x: "8. List All Members");
293             System.out.println(x: "9. Exit");
294             System.out.print(s: "Enter your choice: ");
```

```
279     public static void main(String[] args) {
297             switch (choice) {
298                 case "1":
299                     System.out.print(s: "Enter Book Title: ");
300                     String title = sc.nextLine();
301                     System.out.print(s: "Enter Author: ");
302                     String author = sc.nextLine();
303                     System.out.print(s: "Enter Category: ");
304                     String category = sc.nextLine();
305                     Book b = manager.addBook(title, author, category);
306                     System.out.println("Book added with ID: " + b.getBookId());
307                     break;
308                 case "2":
309                     System.out.print(s: "Enter Member Name: ");
310                     String name = sc.nextLine();
311                     System.out.print(s: "Enter Email: ");
312                     String email = sc.nextLine();
313                     Member m = manager.addMember(name, email);
314                     System.out.println("Member added with ID: " + m.getMemberId());
315                     break;
316                 case "3":
317                     System.out.print(s: "Enter Book ID: ");
318                     int bid = Integer.parseInt(sc.nextLine());
319                     System.out.print(s: "Enter Member ID: ");
320                     int mid = Integer.parseInt(sc.nextLine());
321                     if (manager.issueBook(bid, mid))
322                         System.out.println(x: "Book issued successfully.");
323                     else
324                         System.out.println(x: "Issue failed. Book might already be issued or ID invalid.");
325                     break;
326                 case "4":
327                     System.out.print(s: "Enter Book ID: ");
328                     int rb = Integer.parseInt(sc.nextLine());
329                     System.out.print(s: "Enter Member ID: ");
330                     int rm = Integer.parseInt(sc.nextLine());
331                     if (manager.returnBook(rb, rm))
332                         System.out.println(x: "Book returned successfully.");
333                     else
334                         System.out.println(x: "Return failed. Check IDs or issue status.");
335                     break;
336                 case "5":
337                     System.out.print(s: "Search by (title/author/category): ");
338                     String by = sc.nextLine();
```

```
336                 case "5":
337                     System.out.print(s: "Search by (title/author/category): ");
338                     String by = sc.nextLine();
339                     System.out.print(s: "Enter keyword: ");
340                     String key = sc.nextLine();
341                     List<Book> results = manager.searchBooks(key, by);
342                     if (results.isEmpty())
343                         System.out.println(x: "No results found.");
344                     else
345                         results.forEach(Book::displayBookDetails);
346                     break;
347                 case "6":
348                     System.out.print(s: "Sort by (title/author/category): ");
349                     String sort = sc.nextLine();
350                     List<Book> sorted = manager.sortBooks(sort);
351                     sorted.forEach(Book::displayBookDetails);
352                     break;
353                 case "7":
354                     System.out.println(x: "All Books:");
355                     manager.getAllBooks().forEach(Book::displayBookDetails);
356                     break;
357                 case "8":
358                     System.out.println(x: "All Members:");
359                     manager.getAllMembers().forEach(Member::displayMemberDetails);
360                     break;
361                 case "9":
362                     manager.saveAll();
363                     System.out.println(x: "Data saved. Exiting...");
364                     sc.close();
365                     return;
366                 default:
367                     System.out.println(x: "Invalid choice, try again.");
368             }
369         }
370     }
371 }
372
```

Output:-



Explanation of code:-

City Library Digital Management System — Theory Explanation

Objective

The aim of this project is to design and implement a **digital library management system** that allows the library to:

- Store and manage book and member details
- Issue and return books
- Search and sort books
- Save all data permanently using file handling

It combines **Java I/O (input/output)**, **collections**, and **object-oriented programming** concepts.

Major Components

a) Book Class

- Represents each **book** in the library.
- Contains attributes: bookId, title, author, category, and isIssued.
- Methods:
    - displayBookDetails() → prints book info.
    - markAsIssued() and markAsReturned() → update issue status.
    - Implements **Comparable** to allow sorting by book title.
    - Includes file conversion methods (toDataString() and fromDataString()) to read/write data to files.

b) Member Class

- Represents a **library member**.
- Attributes: memberId, name, email, and issuedBooks (list of book IDs issued to the member).
- Methods:
    - addIssuedBook() / returnIssuedBook() → manage issued books.
    - displayMemberDetails() → shows member info.
    - File conversion methods for saving/loading member data.

c) LibraryManager Class

- Core logic of the program.
- Uses **Collections Framework**:
    - Map<Integer, Book> → stores all books.
    - Map<Integer, Member> → stores all members.
    - Set<String> → stores all book categories.
- Handles:
    - Adding books and members.
    - Issuing and returning books.
    - Searching and sorting.

    - Reading and writing data using **BufferedReader/BufferedWriter**.
- Automatically assigns unique IDs and updates text files.

d) Main Class (Menu Section)

- Provides a **console-based menu** for the user.
- Allows user to choose actions like add, search, issue, return, etc.
- Uses **Scanner** for input and calls LibraryManager methods for execution.
- Keeps running until user selects “Exit”.