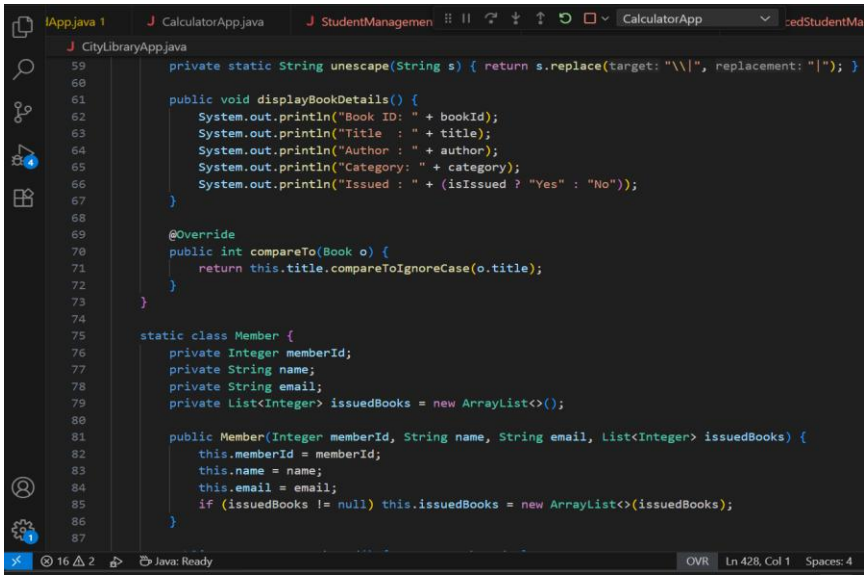
Assignment Number: 04

Name: Aman

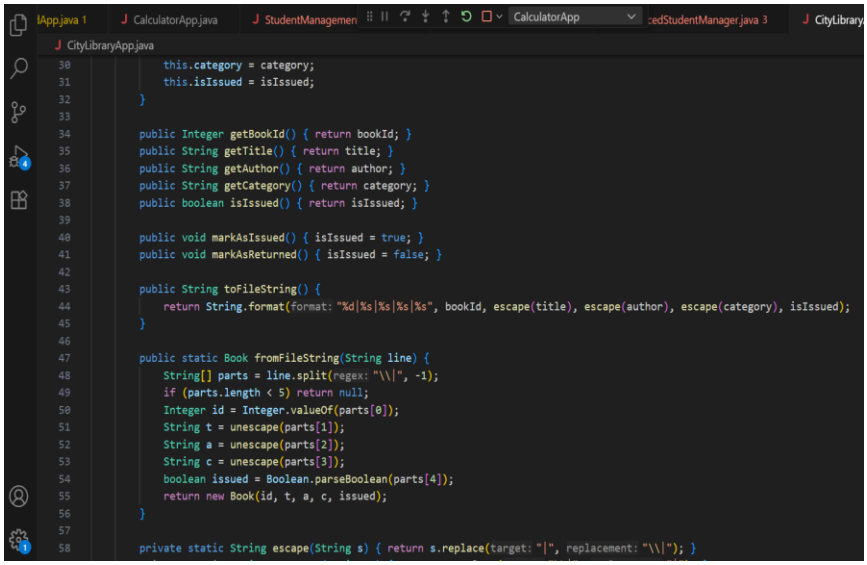
Roll no: 2401201115

Course: BCA (AI&DS)

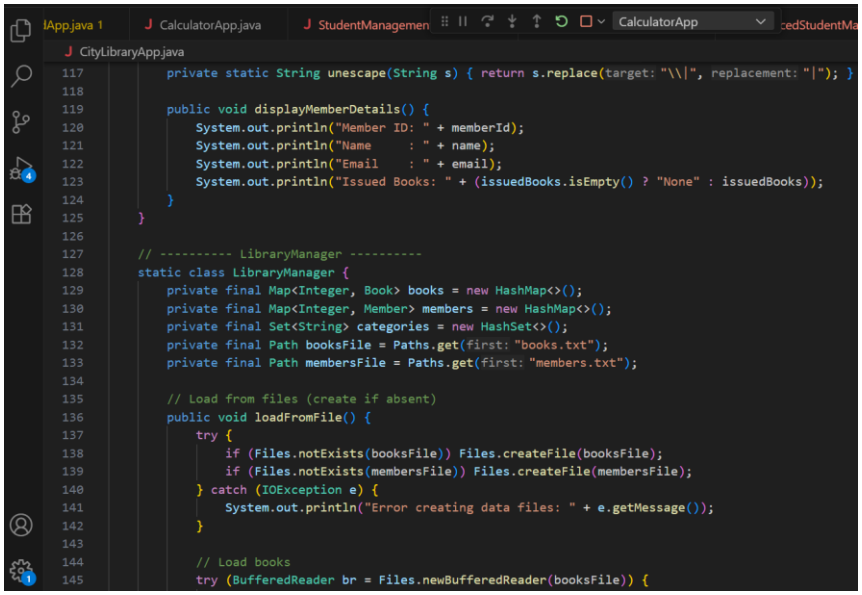
INPUT:

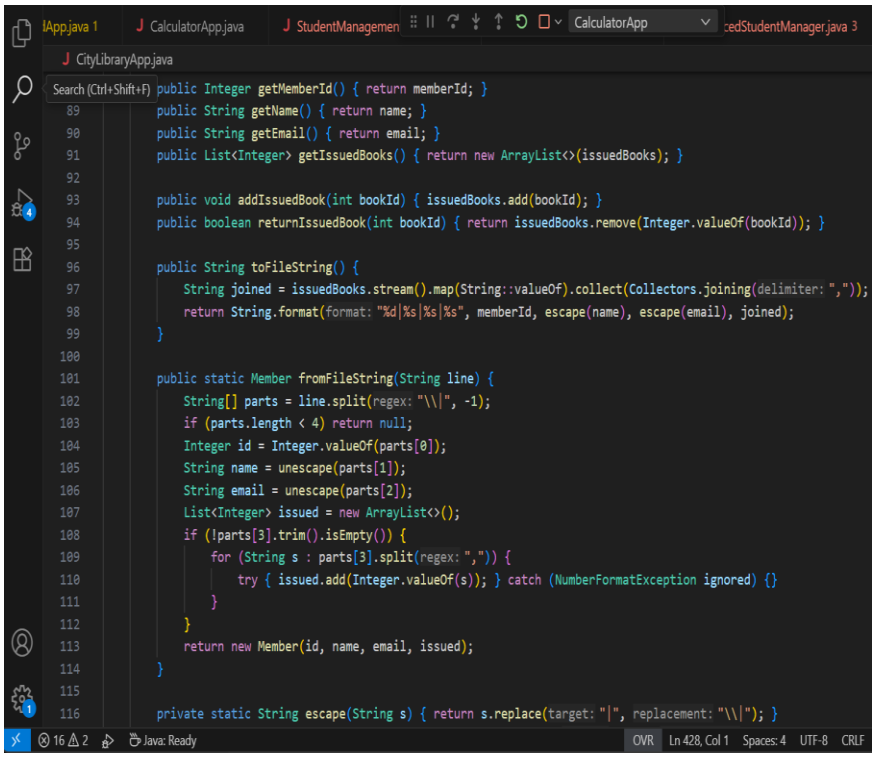


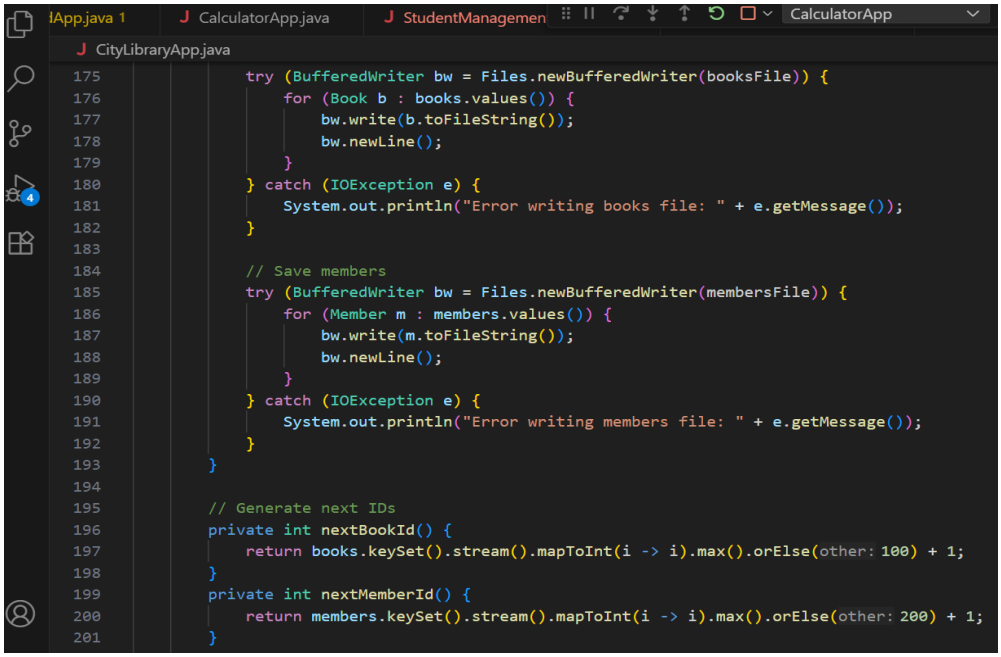
```
        private static String unescape(String s) { return s.replace(target: "\\|", replacement: "|"); }

        public void displayBookDetails() {
            System.out.println("Book ID: " + bookId);
            System.out.println("Title  : " + title);
            System.out.println("Author : " + author);
            System.out.println("Category: " + category);
            System.out.println("Issued : " + (isIssued ? "Yes" : "No"));
        }

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

    static class Member {
        private Integer memberId;
        private String name;
        private String email;
        private List<Integer> issuedBooks = new ArrayList<>();

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

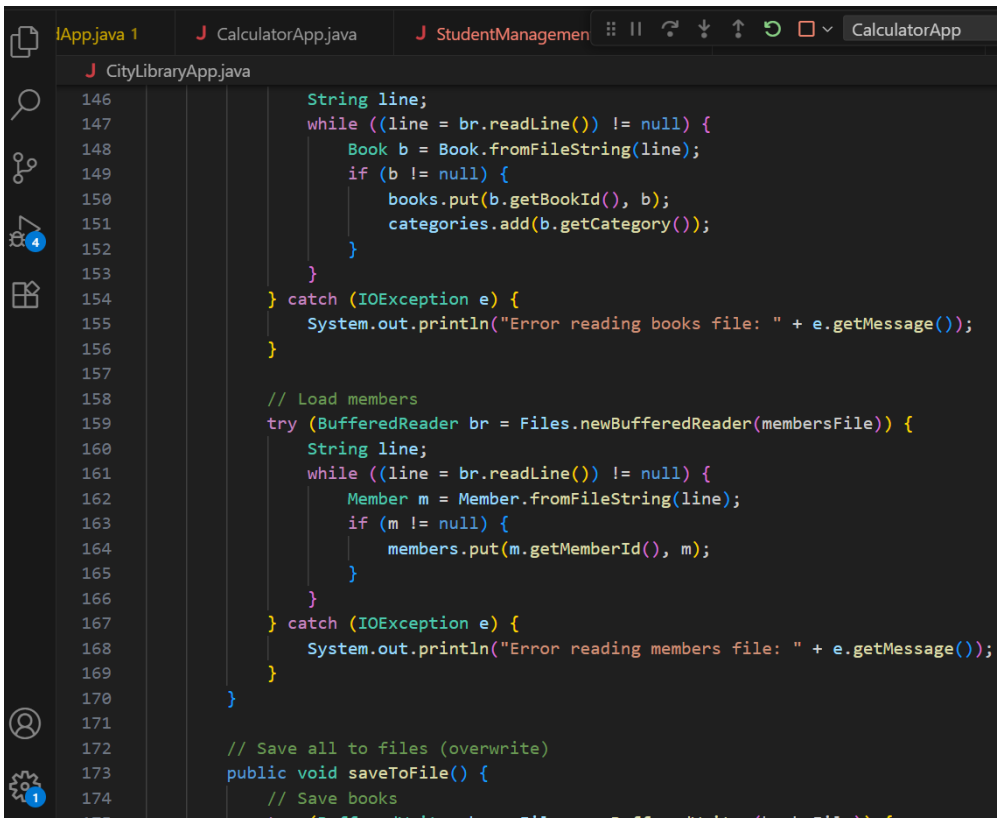


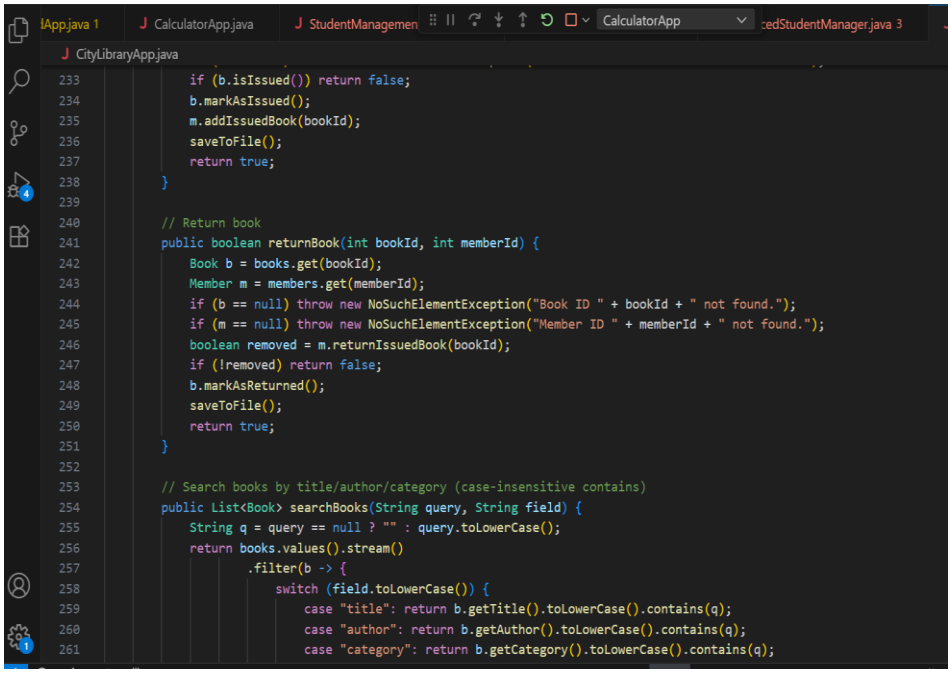
```
            this.category = category;
            this.isIssued = isIssued;
        }

        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; }

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

        public String toFileString() {
            return String.format(format: "%d|%s|%s|%s|%s", bookId, escape(title), escape(author), escape(category), isIssued);
        }

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

        private static String escape(String s) { return s.replace(target: "|", replacement: "\\|"); }
```


CityLibraryApp.java
private static String unescape(String s) { return s.replace(target: "\\|", replacement: "|"); }
public void displayMemberDetails() {
System.out.println("Member ID: " + memberId);
System.out.println("Name : " + name);
System.out.println("Email : " + email);
System.out.println("Issued Books: " + (issuedBooks.isEmpty() ? "None" : issuedBooks));
}
}
// ---------- LibraryManager ----------
static class LibraryManager {
private final Map<Integer, Book> books = new HashMap<>();
private final Map<Integer, Member> members = new HashMap<>();
private final Set<String> categories = new HashSet<>();
private final Path booksFile = Paths.get(first: "books.txt");
private final Path membersFile = Paths.get(first: "members.txt");
// Load from files (create if absent)
public void loadFromFile() {
try {
if (Files.notExists(booksFile)) Files.createFile(booksFile);
if (Files.notExists(membersFile)) Files.createFile(membersFile);
} catch (IOException e) {
System.out.println("Error creating data files: " + e.getMessage());
}
// Load books
try (BufferedReader br = Files.newBufferedReader(booksFile)) {

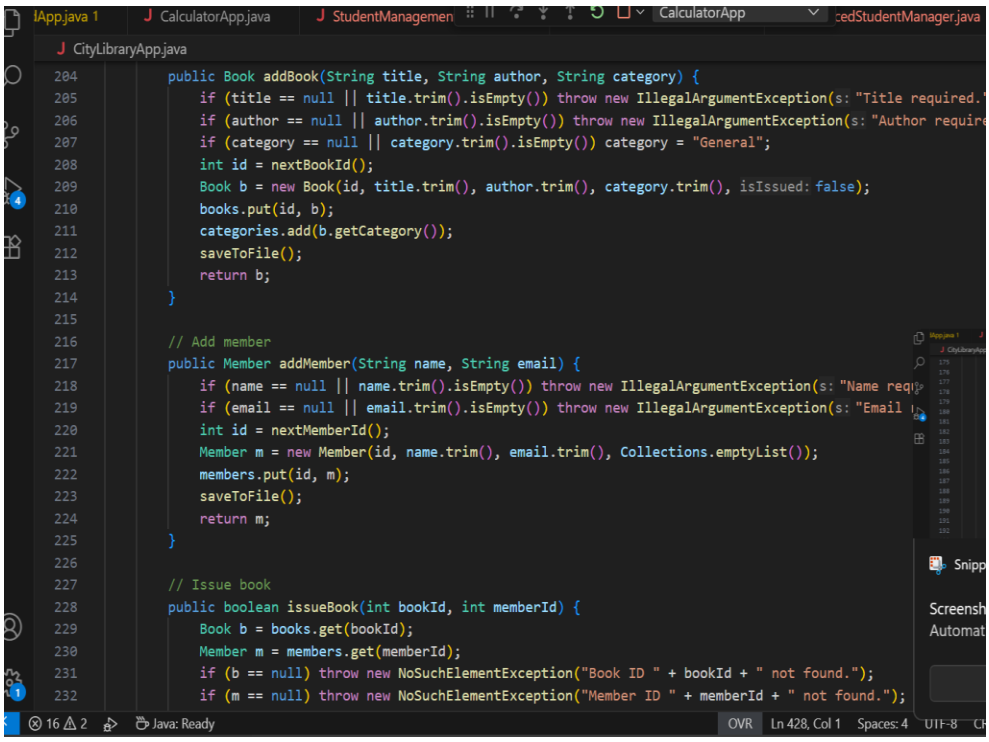

CityLibraryApp.java
Search (Ctrl+Shift+F)
public Integer getMemberId() { return memberId; }
public String getName() { return name; }
public String getEmail() { return email; }
public List<Integer> getIssuedBooks() { return new ArrayList<>(issuedBooks); }
public void addIssuedBook(int bookId) { issuedBooks.add(bookId); }
public boolean returnIssuedBook(int bookId) { return issuedBooks.remove(Integer.valueOf(bookId)); }
public String toFileString() {
String joined = issuedBooks.stream().map(String::valueOf).collect(Collectors.joining(delimiter: ","));
return String.format(format: "%d|%s|%s|%s", memberId, escape(name), escape(email), joined);
}
public static Member fromFileString(String line) {
String[] parts = line.split(regex: "\\|", -1);
if (parts.length < 4) return null;
Integer id = Integer.valueOf(parts[0]);
String name = unescape(parts[1]);
String email = unescape(parts[2]);
List<Integer> issued = new ArrayList<>();
if (!parts[3].trim().isEmpty()) {
for (String s : parts[3].split(regex: ",")) {
try { issued.add(Integer.valueOf(s)); } catch (NumberFormatException ignored) {}
}
}
return new Member(id, name, email, issued);
}
private static String escape(String s) { return s.replace(target: "|", replacement: "\\|"); }
Java: Ready
OVR
Ln 428, Col 1
Spaces: 4
UTF-8
CRLF


IApp.java 1
CalculatorApp.java
StudentManagemen
CalculatorApp
CityLibraryApp.java
try (BufferedWriter bw = Files.newBufferedWriter(booksFile)) {
for (Book b : books.values()) {
bw.write(b.toFileString());
bw.newLine();
}
} catch (IOException e) {
System.out.println("Error writing books file: " + e.getMessage());
}
// Save members
try (BufferedWriter bw = Files.newBufferedWriter(membersFile)) {
for (Member m : members.values()) {
bw.write(m.toFileString());
bw.newLine();
}
} catch (IOException e) {
System.out.println("Error writing members file: " + e.getMessage());
}
}
// Generate next IDs
private int nextBookId() {
return books.keySet().stream().mapToInt(i -> i).max().orElse(other: 100) + 1;
}
private int nextMemberId() {
return members.keySet().stream().mapToInt(i -> i).max().orElse(other: 200) + 1;
}


IApp.java 1
CalculatorApp.java
StudentManagemen
CalculatorApp
CityLibraryApp.java
String line;
while ((line = br.readLine()) != null) {
Book b = Book.fromFileString(line);
if (b != null) {
books.put(b.getBookId(), b);
categories.add(b.getCategory());
}
}
} catch (IOException e) {
System.out.println("Error reading books file: " + e.getMessage());
}
// Load members
try (BufferedReader br = Files.newBufferedReader(membersFile)) {
String line;
while ((line = br.readLine()) != null) {
Member m = Member.fromFileString(line);
if (m != null) {
members.put(m.getMemberId(), m);
}
}
} catch (IOException e) {
System.out.println("Error reading members file: " + e.getMessage());
}
}
// Save all to files (overwrite)
public void saveToFile() {
// Save books



```
233         if (b.isIssued()) return false;
234         b.markAsIssued();
235         m.addIssuedBook(bookId);
236         saveToFile();
237         return true;
238     }
239
240     // Return book
241     public boolean returnBook(int bookId, int memberId) {
242         Book b = books.get(bookId);
243         Member m = members.get(memberId);
244         if (b == null) throw new NoSuchElementException("Book ID " + bookId + " not found.");
245         if (m == null) throw new NoSuchElementException("Member ID " + memberId + " not found.");
246         boolean removed = m.returnIssuedBook(bookId);
247         if (!removed) return false;
248         b.markAsReturned();
249         saveToFile();
250         return true;
251     }
252
253     // Search books by title/author/category (case-insensitive contains)
254     public List<Book> searchBooks(String query, String field) {
255         String q = query == null ? "" : query.toLowerCase();
256         return books.values().stream()
257                 .filter(b -> {
258                     switch (field.toLowerCase()) {
259                         case "title": return b.getTitle().toLowerCase().contains(q);
260                         case "author": return b.getAuthor().toLowerCase().contains(q);
261                         case "category": return b.getCategory().toLowerCase().contains(q);
```



```
204     public Book addBook(String title, String author, String category) {
205         if (title == null || title.trim().isEmpty()) throw new IllegalArgumentException(s: "Title required.
206         if (author == null || author.trim().isEmpty()) throw new IllegalArgumentException(s: "Author require
207         if (category == null || category.trim().isEmpty()) category = "General";
208         int id = nextBookId();
209         Book b = new Book(id, title.trim(), author.trim(), category.trim(), isIssued: false);
210         books.put(id, b);
211         categories.add(b.getCategory());
212         saveToFile();
213         return b;
214     }
215
216     // Add member
217     public Member addMember(String name, String email) {
218         if (name == null || name.trim().isEmpty()) throw new IllegalArgumentException(s: "Name requ
219         if (email == null || email.trim().isEmpty()) throw new IllegalArgumentException(s: "Email
220         int id = nextMemberId();
221         Member m = new Member(id, name.trim(), email.trim(), Collections.emptyList());
222         members.put(id, m);
223         saveToFile();
224         return m;
225     }
226
227     // Issue book
228     public boolean issueBook(int bookId, int memberId) {
229         Book b = books.get(bookId);
230         Member m = members.get(memberId);
231         if (b == null) throw new NoSuchElementException("Book ID " + bookId + " not found.");
232         if (m == null) throw new NoSuchElementException("Member ID " + memberId + " not found.");
```

CityLibraryApp.java

```
293     public static void main(String[] args) {
294         Scanner sc = new Scanner(System.in);
295         LibraryManager manager = new LibraryManager();
296         manager.loadFromFile();
297 
298         boolean running = true;
299         System.out.println(x: "Welcome to City Library Digital Management System");
300         while (running) {
301             try {
302                 System.out.println(x: "\n1. Add Book");
303                 System.out.println(x: "2. Add Member");
304                 System.out.println(x: "3. Issue Book");
305                 System.out.println(x: "4. Return Book");
306                 System.out.println(x: "5. Search Books");
307                 System.out.println(x: "6. Sort Books");
308                 System.out.println(x: "7. View All Books");
309                 System.out.println(x: "8. View All Members");
310                 System.out.println(x: "9. Exit");
311                 System.out.print(s: "Enter your choice: ");
312                 String choiceLine = sc.nextLine().trim();
313                 int choice = Integer.parseInt(choiceLine);
314 
315                 switch (choice) {
316                     case 1 -> {
317                         System.out.print(s: "Enter Book Title: ");
318                         String title = sc.nextLine();
319                         System.out.print(s: "Enter Author: ");
320                         String author = sc.nextLine();
321                         System.out.print(s: "Enter Category: ");
```

IApp.java 1 | CalculatorApp.java | StudentManagemen | CalculatorApp | cedStudentM

CityLibraryApp.java

```
262                         default:
263                             return b.getTitle().toLowerCase().contains(q)
264                                     || b.getAuthor().toLowerCase().contains(q)
265                                     || b.getCategory().toLowerCase().contains(q);
266                     }
267                 })
268                 .collect(Collectors.toList());
269         }
270 
271         // Sort books by title (Comparable) or by author/category via Comparator
272         public List<Book> sortBooks(String by) {
273             List<Book> list = new ArrayList<>(books.values());
274             switch (by.toLowerCase()) {
275                 case "author":
276                     list.sort(Comparator.comparing(Book::getAuthor, String.CASE_INSENSITIVE_ORDER));
277                     break;
278                 case "category":
279                     list.sort(Comparator.comparing(Book::getCategory, String.CASE_INSENSITIVE_ORDER))
280                     break;
281                 default:
282                     Collections.sort(list); // by title (Comparable)
283             }
284             return list;
285         }
286 
287         public Map<Integer, Book> getBooks() { return new HashMap<>(books); }
288         public Map<Integer, Member> getMembers() { return new HashMap<>(members); }
289         public Set<String> getCategories() { return new HashSet<>(categories); }
290     }
```

CityLibraryApp.java

```
351                 System.out.print(s: "Enter search query: ");
352                 String q = sc.nextLine().trim();
353                 List<Book> results = manager.searchBooks(q, field);
354                 if (results.isEmpty()) System.out.println(x: "No books found.");
355                 else {
356                     System.out.println(x: "Search Results:");
357                     for (Book book : results) {
358                         System.out.println(x: "----");
359                         book.displayBookDetails();
360                     }
361                 }
362             }
363             case 6 -> {
364                 System.out.print(s: "Sort by (title/author/category): ");
365                 String by = sc.nextLine().trim();
366                 List<Book> sorted = manager.sortBooks(by);
367                 System.out.println(x: "Sorted Books:");
368                 for (Book book : sorted) {
369                     System.out.println(x: "----");
370                     book.displayBookDetails();
371                 }
372             }
373             case 7 -> {
374                 Map<Integer, Book> all = manager.getBooks();
375                 if (all.isEmpty()) System.out.println(x: "No books in library.");
376                 else {
377                     System.out.println(x: "All Books:");
378                     for (Book b : all.values()) {
379                         System.out.println(x: "----");
```

App.java 1 | CalculatorApp.java | StudentManagemen | CalculatorApp | cedStudentManager.java

CityLibraryApp.java

```
322                 String category = sc.nextLine();
323                 Book b = manager.addBook(title, author, category);
324                 System.out.println("Book added successfully with ID: " + b.getBookId());
    Click to add a breakpoint
326             }
327             case 2 -> {
328                 System.out.print(s: "Enter Member Name: ");
329                 String name = sc.nextLine();
330                 System.out.print(s: "Enter Email: ");
331                 String email = sc.nextLine();
332                 Member m = manager.addMember(name, email);
333                 System.out.println("Member added successfully with ID: " + m.getMemberId());
334             }
335             case 3 -> {
336                 int bookId = readInt(sc, prompt: "Enter Book ID to issue: ");
337                 int memberId = readInt(sc, prompt: "Enter Member ID: ");
338                 boolean ok = manager.issueBook(bookId, memberId);
339                 if (ok) System.out.println("Book " + bookId + " issued to member " + memberId);
340                 else System.out.println(x: "Book is already issued.");
341             }
342             case 4 -> {
343                 int bookId = readInt(sc, prompt: "Enter Book ID to return: ");
344                 int memberId = readInt(sc, prompt: "Enter Member ID: ");
345                 boolean ok = manager.returnBook(bookId, memberId);
346                 if (ok) System.out.println("Book " + bookId + " returned by member " + memberId);
347                 else System.out.println(x: "This member did not have that book issued.");
348             }
349             case 5 -> {
350                 System.out.print(s: "Search field (title/author/category/all): ");
351                 String field = sc.nextLine().trim();
```

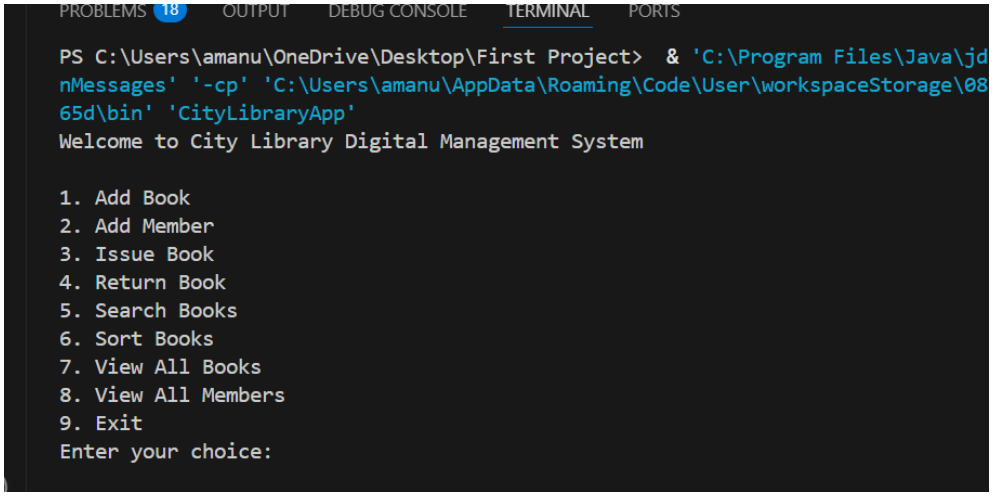
CityLibraryApp.java

```
409                     System.out.println("Unexpected error: " + ex.getMessage());
410                 }
411             }
412
413             sc.close();
414         }
415
416         private static int readInt(Scanner sc, String prompt) {
417             while (true) {
418                 try {
419                     System.out.print(prompt);
420                     String line = sc.nextLine().trim();
421                     return Integer.parseInt(line);
422                 } catch (NumberFormatException e) {
423                     System.out.println(x: "Please enter a valid integer.");
424                 }
425             }
426         }
427     }
```

CityLibraryApp.java

```
380                                 b.displayBookDetails();
381                             }
382                         }
383                     }
384                     case 8 -> {
385                         Map<Integer, Member> all = manager.getMembers();
386                         if (all.isEmpty()) System.out.println(x: "No members.");
387                         else {
388                             System.out.println(x: "All Members:");
389                             for (Member mem : all.values()) {
390                                 System.out.println(x: "----");
391                                 mem.displayMemberDetails();
392                             }
393                         }
394                     }
395                     case 9 -> {
396                         manager.saveToFile();
397                         System.out.println(x: "Data saved. Exiting. Goodbye!");
398                         running = false;
399                     }
400                     default -> System.out.println(x: "Invalid choice. Try again.");
401                 }
402             } catch (NumberFormatException nfe) {
403                 System.out.println(x: "Invalid number input. Try again.");
404             } catch (IllegalArgumentException iae) {
405                 System.out.println("Validation error: " + iae.getMessage());
406             } catch (NoSuchElementException nsee) {
407                 System.out.println("Not found: " + nsee.getMessage());
408             } catch (Exception ex) {
```

OUTPUT:



Explanation —

1 — High-level overview

`CityLibraryApp` is a console-based library management system that lets you add books and members, issue and return books, search and sort books, and persist data between runs. It stores data in two text files (`books.txt`, `members.txt`) and uses the Java Collections Framework (`Map`, `List`, `Set`) for in-memory operations.

---

2 — Main components (what each part does)

Model classes

Book

- Fields: `Integer bookId`, `String title`, `String author`, `String category`, `boolean isIssued`.
- Responsibilities:
  - Represent a book.
  - Serialize/deserialize itself to/from a single-line file format (`toFileString()` / `fromFileString()`).
  - Mark issued/returned (`markAsIssued()`, `markAsReturned()`).
  - Provide `displayBookDetails()` for console output.
  - Implements `Comparable<Book>` to sort by `title` (case-insensitive).

Member

- Fields: `Integer memberId`, `String name`, `String email`, `List<Integer> issuedBooks`.
- Responsibilities:
  - Represent a library member and track issued book IDs.
  - Serialize/deserialize to/from file lines (`toFileString()` / `fromFileString()`).
  - Add/return book IDs and display member info.

Both classes include small helper functions to escape/unescape `|` so the pipe character can be used as a safe field delimiter.

---

3 — LibraryManager (core logic / service)

`LibraryManager` holds the in-memory collections and handles persistence:

- **Collections**
  - `Map<Integer, Book> books` — fast lookup by bookId.
  - `Map<Integer, Member> members` — fast lookup by memberId.
  - `Set<String> categories` — unique book categories.
- **Files**
  - `books.txt` and `members.txt` under the program folder. Created if missing.
- **Key methods**
  - `loadFromFile()` — reads both files at startup, builds `books`, `members`, and `categories`.
  - `saveToFile()` — writes current `books` and `members` back to their files (overwrites).
  - `addBook(title, author, category)` — validates inputs, auto-generates `bookId`, updates collections and files.
  - `addMember(name, email)` — validates inputs, auto-generates `memberId`, updates collections and files.
  - `issueBook(bookId, memberId)` — sets book issued, adds bookId to member's issued list, persists.
  - `returnBook(bookId, memberId)` — removes bookId from member and marks book returned, persists.
  - `searchBooks(query, field)` — case-insensitive filtering by title/author/category (or all fields).
  - `sortBooks(by)` — returns a sorted `List<Book>` using `Comparable` (title) or `Comparator` (author/category).
  - `nextBookId()` / `nextMemberId()` — simple auto-increment (based on max existing id or base value).

Design notes:

- `saveToFile()` is called after any change so files remain in sync.
- `viewAll` getters return copies where appropriate.

---

4 — File format & I/O strategy

- **books.txt**: each line is `bookId|title|author|category|isIssued`
  - Example: `101|Java Programming Mastery|John Smith|Programming|false`
- **members.txt**: each line is `memberId|name|email|issuedBookId1,issuedBookId2`
  - Example: `201|Aman|aman@mail.com|101,105`

I/O uses `java.nio.file.Files` with `BufferedReader`/`BufferedWriter` (via `Files.newBufferedReader` / `newBufferedWriter`). This is efficient and uses try-with-resources pattern, ensuring streams are closed.

---

5 — User interface (main menu)

- A loop printed to console offers options:
  1. Add Book
  2. Add Member
  3. Issue Book
  4. Return Book
  5. Search Books
  6. Sort Books
  7. View All Books
  8. View All Members
  9. Exit
- Inputs are read with `Scanner`. Integer parsing is wrapped in a `try`/`catch` loop where needed (the `readInt` helper re-prompts on invalid integer entry).
- Each menu choice calls into `LibraryManager` methods and prints friendly messages or search/sort results.

---

6 — Error handling & validation

- The code catches and reports:
  - `NumberFormatException` when non-integer input is provided where numeric is expected.
  - `IllegalArgumentException` thrown by `LibraryManager` when input validation fails (e.g., missing title, author, name, or email).
  - `NoSuchElementException` if `issueBook` / `returnBook` attempted for missing book or member.
  - A generic `Exception` catch-all prints unexpected errors.
- Input helpers re-prompt for valid integers to avoid program crashes.

---

7 — Sorting & searching behavior

- `Book` implements `Comparable` to sort by title (default).
- The manager provides `Comparator` usage to sort by `author` or `category`.
- `searchBooks` filters using `Stream` + `filter` + `collect`, comparing lowercase strings for case-insensitive partial matches.

---

8 — Persistence & data safety

- Files are overwritten on every `saveToFile()` call to reflect latest state; this minimizes inconsistency between memory and disk.
- For safer writes in production, you could write to a temp file and atomically rename it (not implemented here).

---

9 — Extensibility & improvements (suggested)

- **Waiting list:** add `Map<Integer, Queue<Integer>>` for popular book waitlists.
- **Non-blocking saves:** write in background thread to keep UI responsive.
- **Serialization:** use `ObjectOutputStream` for binary storage, or switch to a lightweight DB (SQLite) for robust persistence.
- **Input validation:** add email regex validation and more robust escaping for other special chars.
- **Split into packages/files:** separate `model`, `service`, and `ui` for cleaner project structure.

---

10 — Quick summary (one-liner)

`CityLibraryApp` is a practical, single-file library system that demonstrates file-based persistence, collection-based in-memory management, searching/sorting using Comparable/Comparator, and basic input validation — suitable as a functional prototype you can extend into a multi-file project or connect to a database.