

LibraryService.java



Share

```
3 package services;
4
5 import models.Book;
6 import models.Patron;
7 import models.Transaction;
8 import java.io.*;
9 import java.time.LocalDate;
10 import java.util.*;
11 import java.util.stream.Collectors;
12
13 /**
14  * The core service layer handling all business logic,
15  * persistence, and CRUD operations.
16  * This adheres to the modular design principle.
```

LibraryService.java



Share

Run

```
55     }
56 }
57
58 // --- Book Management (FR-1, FR-2, FR-3) ---
59
60 public void addBook(Book book) {
61     if (books.containsKey(book.getIsbn())) {
62         throw new IllegalArgumentException("Book with this
        ISBN already exists.");
63     }
64     books.put(book.getIsbn(), book);
65     saveData();
66 }
67
68 public Book findBook(String query) {
```

LibraryService.java



Share

Run

```
43     }
44 }
45
46 private void saveData() {
47     try (FileOutputStream fos = new FileOutputStream(DATA_FILE
48         );
49         ObjectOutputStream oos = new ObjectOutputStream(fos))
50     {
51         oos.writeObject(books);
52         oos.writeObject(patrons);
53         oos.writeObject(transactions);
54         oos.writeInt(nextTransactionId);
55     } catch (IOException e) {
56         System.err.println("Error saving data: " + e
57             .getMessage());
58     }
59 }
```

LibraryService.java

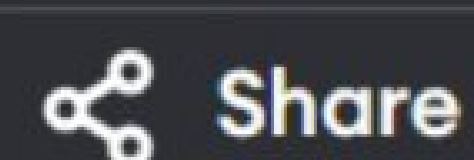
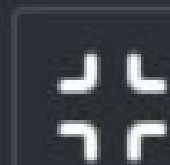


Share

Run

```
30     @SuppressWarnings("unchecked")
31     private void loadData() {
32         try (FileInputStream fis = new FileInputStream(DATA_FILE);
33             ObjectInputStream ois = new ObjectInputStream(fis)) {
34             books = (Map<String, Book>) ois.readObject();
35             patrons = (Map<String, Patron>) ois.readObject();
36             transactions = (List<Transaction>) ois.readObject();
37             nextTransactionId = ois.readInt();
38             System.out.println("Data loaded successfully.");
39         } catch (FileNotFoundException e) {
40             System.out.println("No existing data found. Starting
               fresh.");
41         } catch (IOException | ClassNotFoundException e) {
42             System.err.println("Error loading data: " + e
               .getMessage());
```


LibraryService.java



Run

```
17 public class LibraryService {
18     private static final String DATA_FILE = "library_data.ser"; //
        Persistence
19     private Map<String, Book> books = new HashMap<>();
20     private Map<String, Patron> patrons = new HashMap<>();
21     private List<Transaction> transactions = new ArrayList<>();
22     private int nextTransactionId = 1;
23
24     public LibraryService() {
25         loadData(); // NFR-1: Reliability - Load data on startup
26     }
27
28     // --- Data Persistence (NFR-1: Reliability) ---
29
30     @SuppressWarnings("unchecked")
```

LibraryService.java



Share

Run

```
107     boolean hasActiveLoan = transactions.stream()
108         .filter(t -> t.getPatronId().equals(patronId) &&
            .getBookIsbn().equals(bookIsbn) && !t
            .isReturned())
109         .findAny().isPresent();
110     if (hasActiveLoan) throw new IllegalStateException("Patron
        already has an active loan for this book.");
111
112
113     // Core business logic
114     if (book.checkOut()) {
115         LocalDate borrowDate = LocalDate.now();
116         LocalDate dueDate = borrowDate.plusDays(loanDays);
117         String tId = String.valueOf(nextTransactionId++);
118         Transaction transaction = new Transaction(tId,
```

LibraryService.java




Share

Run

```
97 public void checkOutBook(String bookIsbn, String patronId, int
    loanDays) {
98     Book book = books.get(bookIsbn);
99     Patron patron = patrons.get(patronId);
100
101     // NFR-4: Error Handling & FR-8: Input Validation
102     if (book == null) throw new IllegalArgumentException("Book
        not found.");
103     if (patron == null) throw new IllegalArgumentException
        ("Patron not found.");
104     if (!book.isAvailable()) throw new IllegalStateException
        ("All copies of this book are currently checked out."
        );
105
106     // Check if patron has an active loan for this specific
```

LibraryService.java

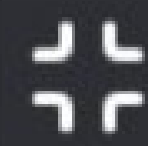


 Share

Run

```
83 public void addPatron(Patron patron) {
84     if (patrons.containsKey(patron.getPatronId())) {
85         throw new IllegalArgumentException("Patron with this
            ID already exists.");
86     }
87     patrons.put(patron.getPatronId(), patron);
88     saveData();
89 }
90
91 public Patron findPatron(String patronId) {
92     return patrons.get(patronId);
93 }
94
95 // --- Borrowing & Return Management (FR-5, FR-6) ---
96
```


LibraryService.java



Share

Run

```
69         return books.values().stream()
70             .filter(b -> b.getIsbn().equalsIgnoreCase(query)
71                 ||
72                 b.getTitle().toLowerCase().contains
73                     (query.toLowerCase()) ||
74                 b.getAuthor().toLowerCase().contains
75                     (query.toLowerCase()))
76             .findFirst()
77             .orElse(null);
78     }
79
80     public Collection<Book> getAllBooks() {
81         return books.values();
82     }
83 }
```

LibraryService.java

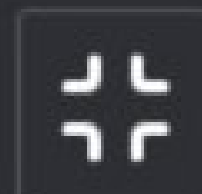


Share

Run

```
118         Transaction transaction = new Transaction(tId,
119             bookIsbn, patronId, borrowDate, dueDate);
120         transactions.add(transaction);
121         saveData();
122         System.out.printf("Successfully checked out! Due Date:
123             %s\n", dueDate);
124     }
125 }
126
127 public double returnBook(String bookIsbn, String patronId) {
128     Book book = books.get(bookIsbn);
129     if (book == null) throw new IllegalArgumentException("Book
    not found.");
130
131     // Find the active transaction for this book/patron pair
```

LibraryService.java



Share

Run

```
130 Optional<Transaction> activeTransaction = transactions
    .stream()
131     .filter(t -> t.getBookIsbn().equals(bookIsbn) && t
        .getPatronId().equals(patronId) && !t
        .isReturned())
132     .findFirst();
133
134 if (activeTransaction.isEmpty()) {
135     throw new IllegalStateException("No active loan found
        for this book and patron combination.");
136 }
137
138 Transaction transaction = activeTransaction.get();
139 transaction.setReturnDate(LocalDate.now());
140 book.checkIn(): // Update book status
```

LibraryService.java



Share

Run

```
143     double fine = 0.0;
144     if (transaction.isOverdue()) {
145         long overdueDays = java.time.temporal.ChronoUnit.DAYS
            .between(transaction.getDueDate(), LocalDate.now
                ());
146         fine = overdueDays * 0.50; // $0.50 fine per overdue
            day
147     }
148
149     saveData();
150     return fine;
151 }
152
153 // --- Reporting (FR-7) ---
154
```


LibraryService.java



Share

Run

```
152
153     // --- Reporting (FR-7) ---
154
155     public List<Transaction> getPatronHistory(String patronId) {
156         return transactions.stream()
157             .filter(t -> t.getPatronId().equals(patronId))
158             .collect(Collectors.toList());
159     }
160
161     public List<Transaction> getOverdueLoans() {
162         return transactions.stream()
163             .filter(Transaction::isOverdue)
164             .collect(Collectors.toList());
165     }
166 }
```