

LibraryService.java



```
- 1 package services;
 2
 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.
17 */
```

LibraryService.java

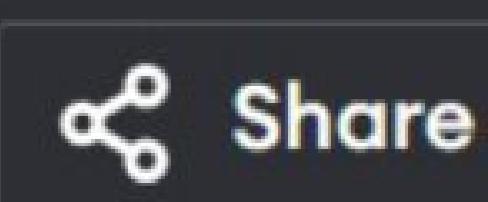


Share

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
63             ISBN already exists.");
64     }
65     books.put(book.getIsbn(), book);
66     saveData();
67 }
68
69 public Book findBook(String query) {
```

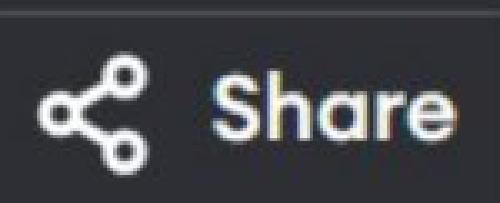
LibraryService.java



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         }
60     }
61 }
```

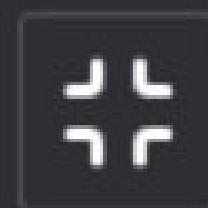
LibraryService.java



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
41                             fresh.");
42         } catch (IOException | ClassNotFoundException e) {
43             System.err.println("Error loading data: " + e
44                               .getMessage());
```

LibraryService.java

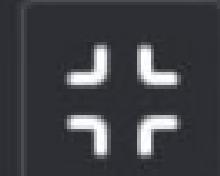


Share

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) &&
109             .getBookIsbn().equals(bookIsbn) && !t
110             .isReturned())
111         .findAny().isPresent();
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

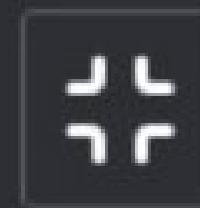


Share

Run

```
97     public void checkOutBook(String bookIsbn, String patronId, int
98         loanDays) {
99             Book book = books.get(bookIsbn);
100            Patron patron = patrons.get(patronId);
101
102            // NFR-4: Error Handling & FR-8: Input Validation
103            if (book == null) throw new IllegalArgumentException("Book
104                not found.");
105            if (patron == null) throw new IllegalArgumentException
106                ("Patron not found.");
107            if (!book.isAvailable()) throw new IllegalStateException
108                ("All copies of this book are currently checked out."
109            );
110
111            // Check if patron has an active loan for this specific
112            // book. If so, update the loan's due date and mark the book
113            // as unavailable. If not, add a new loan record.
```

LibraryService.java

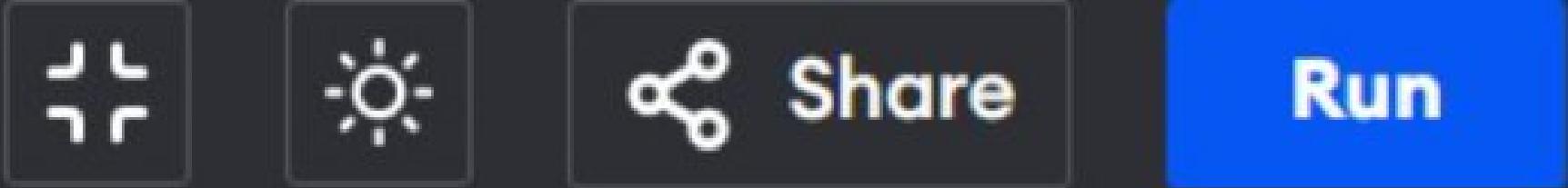


Share

Run

```
83     public void addPatron(Patron patron) {  
84         if (patrons.containsKey(patron.getPatronId())) {  
85             throw new IllegalArgumentException("Patron with this  
86                 ID already exists.");  
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



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

LibraryService.java

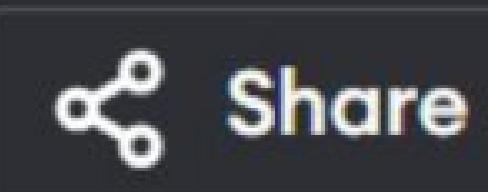
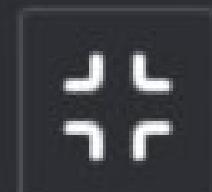


Share

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     public double returnBook(String bookIsbn, String patronId) {
126         Book book = books.get(bookIsbn);
127         if (book == null) throw new IllegalArgumentException("Book
128             not found.");
129         // Find the active transaction for this book/patron pair
```

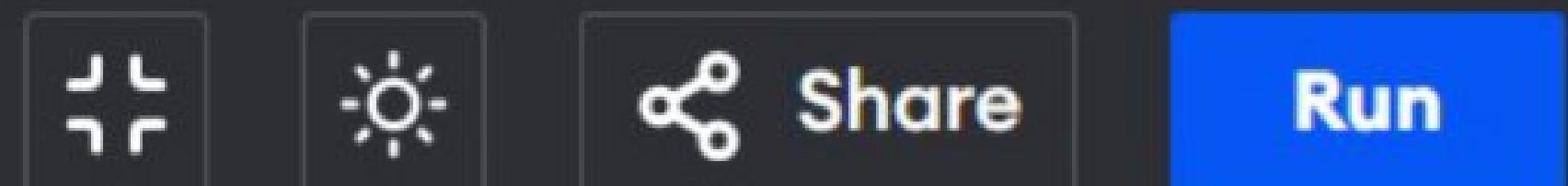
LibraryService.java



Run

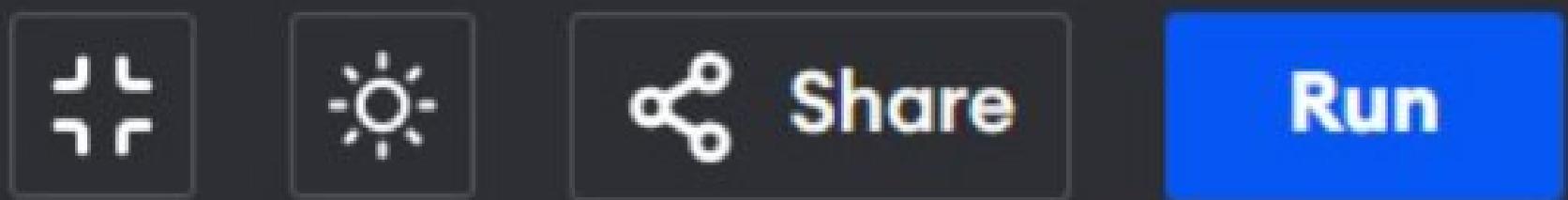
```
130     Optional<Transaction> activeTransaction = transactions
131         .stream()
132             .filter(t -> t.getBookIsbn().equals(bookIsbn) && t
133                 .getPatronId().equals(patronId) && !t
134                 .isReturned())
135             .findFirst();
136
137
138     Transaction transaction = activeTransaction.get();
139     transaction.setReturnDate(LocalDate.now());
140     book.checkIn(); // Update book status
```

LibraryService.java



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

LibraryService.java



```
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 }
```