ROLLINS LIBRARY MANAGEMENT APPLICATION

Joshua Mitchell: Frontend Developer
David Austen: Backend Developer
Taylor Adams: Documenter/Management

Introduction

Within the assignment, we got together and decided to create a library management system. We believe this project would be great for us because it involves many core concepts of object-oriented programming, such as inheritance, encapsulation, abstraction, GUI principles, and polymorphism. We also believed that creating a library system was fitting for a project associated with Rollins College. We were able to utilize Rollins' personal library archive as a reference and resource for data within our own library archive. As a collective, we also thought that a project like this would be challenging, yet familiar with our own experience with the blend of object-oriented programming principles.

Implementation Description

Our Rollins College Library Management System was built using basic Object-Oriented Programming principles. This method helped us create a modular, scalable, and easy-to-maintain application. Below is a breakdown of how we applied each principle.

Encapsulation

Encapsulation was key in our design. We combined the data (attributes) and the methods (behaviors) that operate on the data into single units, or classes. Importantly, we only allowed access to an object's internal state through public methods, protecting it from unintended outside interference.

Inheritance

We mainly applied polymorphism through two methods: method overriding and using generic collections.

Examples:
Method Overriding (toString): All our entity classes override the toString() method inherited from the root Object class. This allows each object to define its own string representation. When methods like JTextArea.append(Object obj) call toString(), the appropriate version specific to each class is executed.

Polymorphism in Collections: The loans list in LibraryUI contains objects of type LoanBook. However, it interacts seamlessly with the Book and Student objects within each LoanBook. When we call loan.getBook().getTitle(), we work with the Book object at runtime without knowing its exact type, showing polymorphism in action.

Abstraction

We used abstraction to hide complex implementation details and reveal only essential features. Our classes represent abstractions for real-world concepts.

Examples:
Data Management Abstraction: The LibraryDataManager class is a prime example. It offers two straightforward methods: saveData and loadData. LibraryUI does not need to know about file I/O, parsing, or the specific data format (#BOOKS, #STUDENTS sections). It simply calls these methods, hiding the complexity.

Challenges and Solutions

While developing the Rollins College Library Management System, our team faced significant challenges. Overcoming them needed a mix of research, teamwork, and problem-solving. One major challenge was designing and building the LibraryUI class to be both functional and visually appealing. Creating a complex Swing-based GUI with multiple tabs, a unified color scheme, and a logical layout was a tough task. Aligning components using GridBagLayout, managing CardLayout for switching between login and the main application, and ensuring the UI was user-friendly required careful effort. We took a multi-faceted approach to address this challenge:

Leveraging Past Work: We revisited code from previous assignments and labs that used Swing components. This provided foundational knowledge and reusable code snippets for panels, buttons, and layout managers.

Targeted Research: We conducted focused research on Java Swing libraries and best practices. Official Oracle documentation, Stack Overflow, and Java tutorials proved helpful for learning advanced techniques, like using JTabbedPane for organization and BorderFactory for creating consistent borders.

Incremental Development: Rather than building the entire UI at once, we developed it gradually. We started with a basic login panel, ensured it functioned well, and then created individual tabs one by one (Books, Loan, Return, etc.). This modular approach simplified debugging.

Consistent Theming: To achieve a professional look, we defined a color scheme based on Rollins College colors (blue and yellow) and applied it consistently to all UI components. This created a branded and cohesive user experience.

Challenge 2: Managing Data Persistence and Object Relationships
Description:
A key requirement was for the application to remember its state between sessions. This meant all books, students, and active loans had to be saved to a file and reloaded. The challenge was twofold: first, designing a file format to represent this interrelated data, and second, correctly

reconstructing the network of objects (Book, Student, LoanBook) when loading, including re-linking loans to their corresponding books and students.

Solution:
We created the LibraryDataManager utility class to centralize all file I/O operations.

Custom File Format: We designed a simple, pipe-delimited text format with section headers (#BOOKS, #STUDENTS, #LOANS). This made the file human-readable and easy to parse.

Saving Relationships by ID: We saved the unique identifiers (IDs) that define relationships instead of saving entire objects. For example, a loan entry in the file is stored as BookID->StudentID->LoanDate.

Re-linking on Load: In the loadData method, after loading all books and students into their lists, we process the loans section. For each loan record, we look for matching IDs in the books and students' lists. When found, we create a new LoanBook object to re-establish the link. This successfully restored the system's state perfectly upon startup.

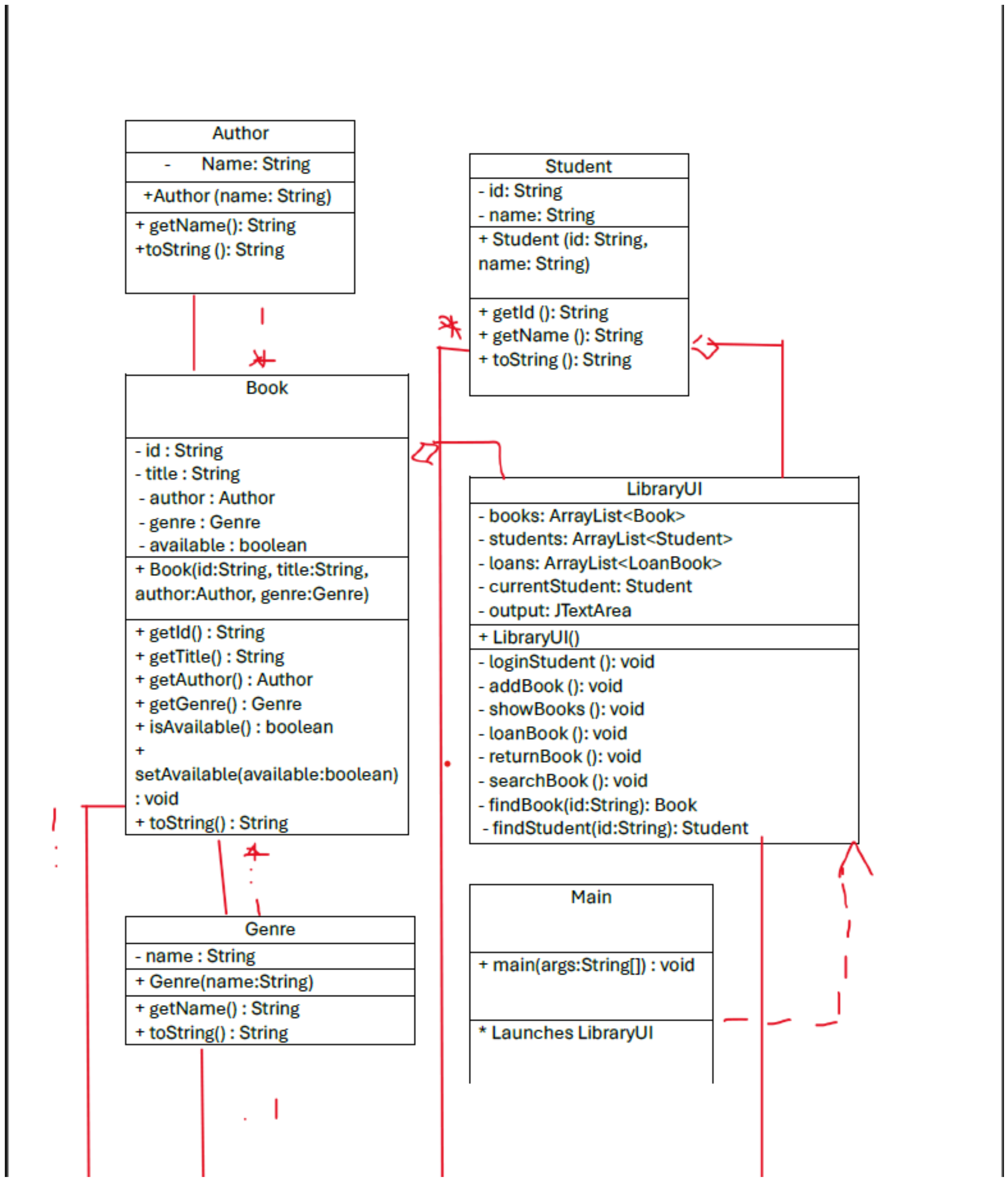
Contribution Summary

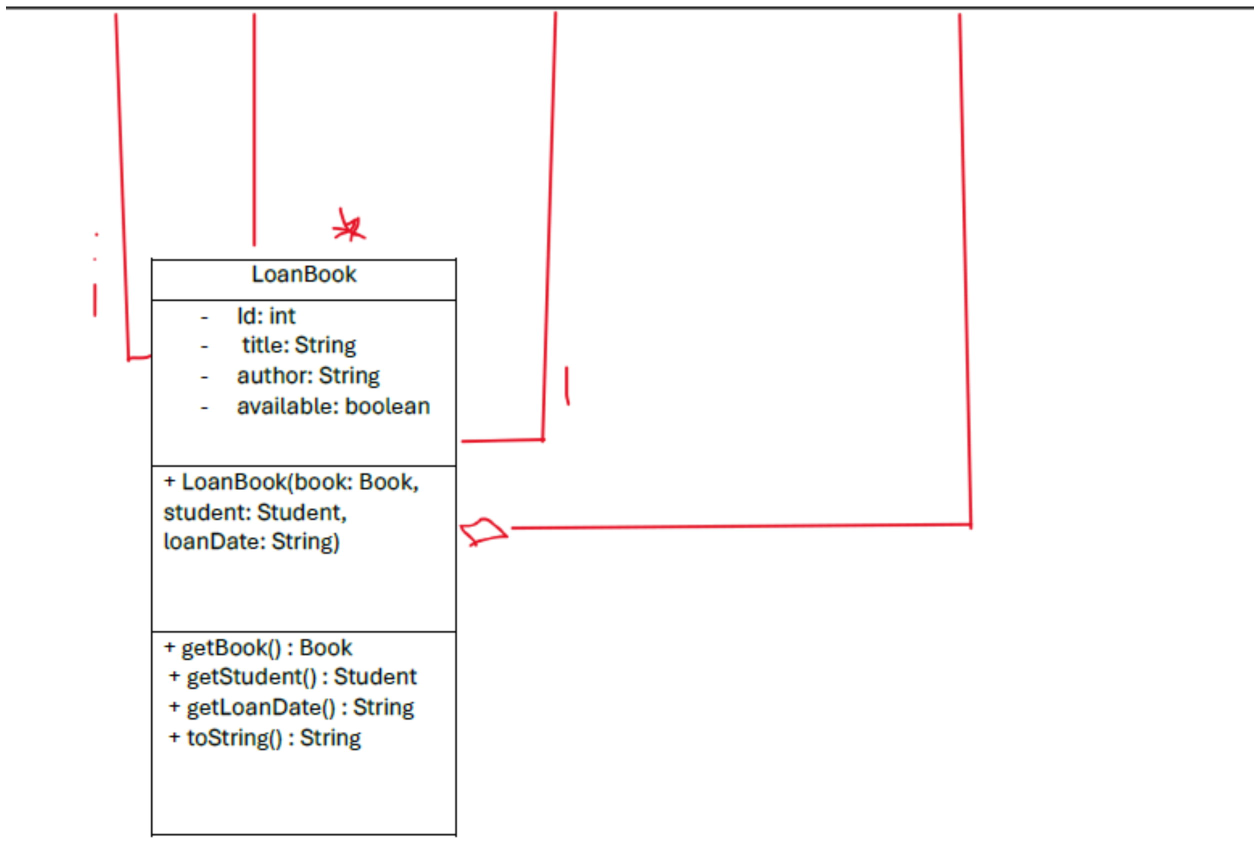
Joshua Mitchell: My main focus within the project was dealing with the frontend aspect of our library application. The root of my job was interlinking the other classes with the LibraryUI class. I was also in charge of the design and paneling of the application. I took a lot of time working with the javax.swing library to find what aesthetically fits best for our project.

David Austen: I dealt with the backend portion working on how each book is saved and added along with what books were pre loaded into the Library System. I created the 8 classes but most of my work was within the Genre, Author, Book, Student, and Main classes.

Taylor Adams: I dealt with all of the documentation and management of the project such as making sure our code and our UML Diagram was implemented and working properly and went over any edits, corrections, and debugging strategies to make sure everything was running smoothly and adequately. I also dealt with commenting on our code to detail what part does what and how everything works in the end.

UML Chart


Author
- Name: String
+Author (name: String)
+ getName(): String
+toString (): String
Student
- id: String
- name: String
+ Student (id: String, name: String)
+ getId (): String
+ getName (): String
+ toString (): String
Book
- id : String
- title : String
- author : Author
- genre : Genre
- available : boolean
+ Book(id:String, title:String, author:Author, genre:Genre)
+ getId() : String
+ getTitle() : String
+ getAuthor() : Author
+ getGenre() : Genre
+ isAvailable() : boolean
+ setAvailable(available:boolean) : void
+ toString() : String
LibraryUI
- books: ArrayList<Book>
- students: ArrayList<Student>
- loans: ArrayList<LoanBook>
- currentStudent: Student
- output: JTextArea
+ LibraryUI()
- loginStudent (): void
- addBook (): void
- showBooks (): void
- loanBook (): void
- returnBook (): void
- searchBook (): void
- findBook(id:String): Book
- findStudent(id:String): Student
Genre
- name : String
+ Genre(name:String)
+ getName() : String
+ toString() : String
Main
+ main(args:String[]) : void
* Launches LibraryUI


*
1
1
LoanBook
- Id: int
- title: String
- author: String
- available: boolean
+ LoanBook(book: Book, student: Student, loanDate: String)
+ getBook() : Book
+ getStudent() : Student
+ getLoanDate() : String
+ toString() : String