**Design Document Modular School Management System**

**Focus Areas:** Fee Tracking and Library Book Management  
**Student Name:** Ian Gitonga  
**Course:** Data Structures and Algorithms  
**Date:** October 2025

**1. Introduction**

This document presents the design and implementation of a **modular School Management System** focusing on two critical administrative modules:

1. **Fee Tracking and Reporting**
2. **Library Book Management**

Both modules are implemented in **Java**, emphasizing the practical use of **data structures** such as the **Binary Search Tree (BST)** for the Fee Module and **HashMap + Stack** for the Library Module.

.The system demonstrates how classical data structures can efficiently manage school records, payments, and library operations while maintaining ethical and performance standards.

**2.System Architecture.**

**Goals**

**1.Modularity.** Each subsystem functions independently.

1. **Scalability.** New modules (attendance, exams) can be added easily.

**3.Maintainability:** Each class is simple and reusable.

**4.Transparency.** Reports are accurate and easy to generate.

* The **Fee Manager** handles financial operations using a **BST**.
* The **Library Manager** uses a **HashMap** for cataloging and a **Stack** for undo operations.

**3. Data Model**

**Fee Tracking Module**

| **Field** | **Data Type** | **Description** |
| --- | --- | --- |
| Student Id | String | Unique identifier for student |
| name | String | Student name |
| Total Fee | double | Total payable amount |
| Amount Paid | double | Amount paid |
| left/right | Fee Node | BST pointers |

**Library Module**

| **Field** | **Data Type** | **Description** |
| --- | --- | --- |
| id | String | Book ID |
| title | String | Book title |
| Is Available | boolean | True if available |

**Entity Relationship Summary**

1.Each student corresponds to one fee record in the BST.  
2.Each book in the library is stored uniquely in a HashMap keyed by its ID.  
3.Stack operations maintain recent book returnhistory for undo actions.

**4. Data Structure Justification**

**Binary Search Tree (Fee Tracking)**

1.Stores student records in sorted order by ID.

2.Allows fast insertion, search, and update in average.

3.Enables range-based reports.

4.Trade-off: May become unbalanced in sorted insertions

**HashMap (Library Management)**

1.Offers average lookup time by book ID.

2.Ideal for **large catalogs** with frequent add/search operations.

3.Handles data with minimal collisions when hashing is efficient.

**Stack (Undo System)**

1.Follows **LIFO** (Last-In-First-Out) structure.

2.Allows easy implementation of undo last return operation.

3.Stores recent actions for borrowing history.

**5. Algorithmic Design and Flow**

**Fee Tracking Algorithms**

**Add Fee Record**

1.If tree is empty → create new node

Else:

2. if ID < current node → go left

3.if ID > current node → go right

4.if ID exists → update payment

**Search Student**

Start at root

If ID matches → return record

Else move left/right depending on ID comparison

**Display All Records**

In-order traversal to print data sorted by student ID

**Library Management Algorithms**

**Add Book**

Insert into HashMap with ID as key

**Borrow Book**

If book available → mark borrowed

Else → print "Book not available"

**Return Book**

Mark available and push book to Stack

**Undo Last Return**

If stack not empty → pop and mark as borrowed again

**6. Binary Search Tree (BST)**

* Organizes nodes hierarchically based on key comparisons.
* Provides average search, insertion, and deletion.
* Supports **sorted traversal** for ordered outputs.

**HashMap**

* Uses **hash functions** to map keys to array indices.
* Provides **constant-time** average operations.
* Handles collisions through **chaining or rehashing**.

**6. Stack**

* LIFO structure allowing **push**, **pop**, and **peek**.
* Supports recursive and undo operations.
* Efficient with time complexity for operations.

**7. Performance Analysis**

**1.BST** provides sorted outputs and good efficiency for moderate datasets.

**2.HashMap** ensures instant access for large libraries.

**3.Stack** allows undo with constant time efficiency.

Trade-off: BST provides order, HashMap provides speed.

**8. Design Decisions**

| **Decision** | **Benefit** | **Cost** |
| --- | --- | --- |
| BST for Fee Tracking | Sorted data and report generation | Risk of unbalanced structure |
| HashMap for Library | O(1) lookup | Memory overhead |
| Stack for Undo | Quick rollback | Limited to recent actions |
| Java language | Object-oriented and portable | Verbose syntax |

**8. Ethical Reflection**

**1.Fairness**

* All students and library users are treated equally.
* Reports and data are generated objectively, without discrimination.

**2 Privacy**

* Personal data (student IDs, payment amounts) must remain confidential.
* System access should be restricted to authorized staff.
* No unnecessary exposure of sensitive information in reports.

**3 Data Protection**

* Backups and encrypted storage recommended for persistence.
* Future upgrades can include user authentication and access logs.

**4 Transparency**

* Fee payments and borrow actions can be traced via system logs.
* Error reporting and accountability are built into design principles.

**5 Ethical Technology Use**

* Designed for educational improvement, not surveillance.
* Encourages responsible, unbiased use of digital records.

**9. Conclusion**

This project demonstrates how fundamental data structures can solve real-world problems effectively and ethically.

The **Fee Tracking System** (BST) provides ordered and efficient financial management.  
The **Library Management System** (HashMap + Stack) ensures rapid retrieval and reversible actions.

Future improvements may include:

1.Attendance tracking using **Queue**

2.Grade analysis using **Graph**

3.Secure file storage and user authentication

Through thoughtful design and ethical awareness, this system exemplifies how **data structures power intelligent and fair educational software**.