**School Management System — Complete Design Document**

**Course:** Data Structures and Algorithms

## Architecture Overview

The School Management System is modular. Each module handles a specific responsibility, and the **main program (SchoolManagementSystem.java)** coordinates interactions.

**Modules and Interaction:**

1. **StudentRegistry.java**
   * Stores student information.
   * Provides data to CourseScheduler and FeeTracker.
2. **CourseScheduler.java**
   * Allocates students to courses in registration order.
   * Receives student objects from StudentRegistry.
3. **FeeTracker.java**
   * Records fee payments.
   * Uses student data from StudentRegistry.
4. **LibrarySystem.java**
   * Manages book borrowing and returns.
   * Receives student info for borrow/return tracking.
5. **PerformanceAnalytics.java**
   * Tracks student scores and identifies top performers.
   * Receives student and course info from StudentRegistry and CourseScheduler.

The main program orchestrates the flow, calling module methods sequentially for operations.

## Data Structure Justification

* **StudentRegistry — HashMap**
  + Fast lookup, add, and remove by student ID.
  + Time Complexity: O(1)
  + Reason: Allows quick access when enrolling in courses or checking fees.
* **CourseScheduler — Queue**
  + Maintains registration order.
  + Time Complexity: O(1) enqueue/dequeue
  + Reason: Ensures fair first-come-first-served course allocation.
* **FeeTracker — Binary Search Tree**
  + Keeps payments sorted by student ID.
  + Time Complexity: O(log n) insert/search, O(n) traversal
  + Reason: Automatically sorted, easy for reporting.
* **LibrarySystem — Stack + HashMap**
  + Stack for borrow/return history; HashMap for quick book lookup.
  + Time Complexity: O(1) for borrow/return/lookup
  + Reason: Efficient tracking and history maintenance.
* **PerformanceAnalytics — Heap/Graph**
  + Heap for top performers; Graph for subject dependencies.
  + Time Complexity: Heap insert O(log n), top O(1); Graph traversal O(V+E)
  + Reason: Efficient top performer detection and visualization of subject relationships.

## Flow Pseudocode

### Student Registration & Course Enrollment

function addStudent(studentID, studentName):  
 student = new Student(studentID, studentName)  
 StudentRegistry.add(student)  
 print(studentName + " registered successfully")  
  
function enrollStudent(student):  
 CourseScheduler.enqueue(student)  
 CourseScheduler.allocateCourses()  
 print(student.name + " allocated to course")

### Fee Tracking

function addPayment(studentID, amount):  
 payment = new FeeNode(studentID, amount)  
 FeeTracker.add(payment)  
 print("Payment recorded for " + studentID)  
  
function showAllPayments():  
 FeeTracker.traverseInOrder()  
 print studentID and amount

### Library Management

function borrowBook(student, ISBN):  
 if ISBN available in HashMap:  
 push ISBN onto Stack  
 mark ISBN as borrowed  
 print("Borrowed: " + ISBN)  
  
function returnBook():  
 ISBN = pop Stack  
 mark ISBN as available  
 print("Returned: " + ISBN)

### Performance Analytics

function addPerformanceRecord(student, subject, score):  
 record = new PerformanceRecord(student, subject, score)  
 PerformanceAnalytics.add(record)  
  
function topPerformer():  
 return Heap.peek()  
  
function showRecords():  
 PerformanceAnalytics.displayAll()