School Management System — Java Code Implementation

This document contains the complete Java source code files for the School Management System prototype. Each section shows the filename, a short description of its purpose, and the full source code. Save each class into a separate .java file with the exact filename shown (or place them in one file as instructed).

## Student.java

Represents a student and stores personal info and course grades.

import java.util.\*;  
  
public class Student {  
 String id;  
 String firstName;  
 String lastName;  
 Map<String, Double> grades; // courseID -> grade  
 boolean feeCleared;  
  
 public Student(String id, String firstName, String lastName) {  
 this.id = id;  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.grades = new HashMap<>();  
 this.feeCleared = false;  
 }  
  
 public double getGPA() {  
 if (grades.isEmpty()) return 0.0;  
 double sum = 0;  
 for (double g : grades.values()) sum += g;  
 return sum / grades.size();  
 }  
  
 @Override  
 public String toString() {  
 return id + " - " + firstName + " " + lastName + " (GPA: " + String.format("%.2f", getGPA()) + ")";  
 }  
}

## Course.java

Models a course: capacity, enrolled students and a waitlist.

import java.util.\*;  
  
public class Course {  
 String id;  
 String title;  
 int capacity;  
 Set<String> enrolled;  
 Queue<String> waitlist;  
  
 public Course(String id, String title, int capacity) {  
 this.id = id;  
 this.title = title;  
 this.capacity = capacity;  
 this.enrolled = new LinkedHashSet<>();  
 this.waitlist = new LinkedList<>();  
 }  
  
 public boolean addStudent(String studentId) {  
 if (enrolled.size() < capacity) {  
 enrolled.add(studentId);  
 return true;  
 } else {  
 waitlist.offer(studentId);  
 return false;  
 }  
 }  
  
 public void dropStudent(String studentId) {  
 enrolled.remove(studentId);  
 if (!waitlist.isEmpty()) {  
 String next = waitlist.poll();  
 enrolled.add(next);  
 }  
 }  
  
 @Override  
 public String toString() {  
 return id + " - " + title + " (cap: " + capacity + ", enrolled: " + enrolled.size() + ")";  
 }  
}

## CircularQueue.java

A simple circular array-backed queue for RegistrationRequest objects.

public class CircularQueue<T> {  
 private T[] data;  
 private int head = 0, tail = 0, size = 0, capacity;  
  
 @SuppressWarnings("unchecked")  
 public CircularQueue(int capacity) {  
 this.capacity = Math.max(4, capacity);  
 this.data = (T[]) new Object[this.capacity];  
 }  
  
 public boolean enqueue(T item) {  
 if (size == capacity) return false; // full  
 data[tail] = item;  
 tail = (tail + 1) % capacity;  
 size++;  
 return true;  
 }  
  
 public T dequeue() {  
 if (size == 0) return null;  
 T item = data[head];  
 data[head] = null;  
 head = (head + 1) % capacity;  
 size--;  
 return item;  
 }  
  
 public boolean isEmpty() {  
 return size == 0;  
 }  
}

## RegistrationRequest.java

Represents a student's request to register for a course (queued).

public class RegistrationRequest {  
 String studentId;  
 String courseId;  
 long requestTime;  
  
 public RegistrationRequest(String studentId, String courseId) {  
 this.studentId = studentId;  
 this.courseId = courseId;  
 this.requestTime = System.currentTimeMillis();  
 }  
}

## FeeRecord.java

Fee transaction record with simple timestamp.

import java.time.\*;  
  
public class FeeRecord {  
 String transactionId;  
 String studentId;  
 double amount;  
 LocalDate date;  
  
 public FeeRecord(String transactionId, String studentId, double amount) {  
 this.transactionId = transactionId;  
 this.studentId = studentId;  
 this.amount = amount;  
 this.date = LocalDate.now();  
 }  
  
 @Override  
 public String toString() {  
 return transactionId + ": " + studentId + " paid " + amount + " on " + date.toString();  
 }  
}

## Book.java

Library book model with copy counts.

public class Book {  
 String isbn;  
 String title;  
 int totalCopies;  
 int availableCopies;  
  
 public Book(String isbn, String title, int copies) {  
 this.isbn = isbn;  
 this.title = title;  
 this.totalCopies = copies;  
 this.availableCopies = copies;  
 }  
  
 @Override  
 public String toString() {  
 return isbn + " - " + title + " (avail: " + availableCopies + ")";  
 }  
}

## StudentPerformance.java

Wrapper to store student id and score for priority queue (comparable).

public class StudentPerformance implements Comparable<StudentPerformance> {  
 String studentId;  
 double score; // e.g., GPA or aggregated score  
  
 public StudentPerformance(String studentId, double score) {  
 this.studentId = studentId;  
 this.score = score;  
 }  
  
 @Override  
 public int compareTo(StudentPerformance other) {  
 // max-heap behavior when used with PriorityQueue by reversing order in comparator  
 return Double.compare(other.score, this.score);  
 }  
  
 @Override  
 public String toString() {  
 return studentId + " → " + String.format("%.2f", score);  
 }  
}

## SchoolController.java

Main controller coordinating modules: registry, courses, fees, library, analytics.

import java.util.\*;  
import java.util.concurrent.atomic.AtomicInteger;  
  
public class SchoolController {  
 // Student registry: HashMap  
 Map<String, Student> studentRegistry = new HashMap<>();  
  
 // Courses: HashMap  
 Map<String, Course> courses = new HashMap<>();  
  
 // Course registration queue: CircularQueue  
 CircularQueue<RegistrationRequest> registrationQueue = new CircularQueue<>(50);  
  
 // Fee tracking: TreeMap (sorted)  
 TreeMap<String, FeeRecord> feeRecords = new TreeMap<>();  
  
 // Library: HashMap + Stack  
 Map<String, Book> books = new HashMap<>();  
 Stack<String> borrowStack = new Stack<>(); // records: isbn:studentId:date  
  
 // Performance: PriorityQueue + adjacency matrix for subject correlation (small demo)  
 PriorityQueue<StudentPerformance> performancePQ = new PriorityQueue<>();  
 // For simplicity, let subjects be indexed 0..n-1  
 double[][] subjectCorrelation; // sample  
  
 AtomicInteger txnCounter = new AtomicInteger(1000);  
  
 // --- Student operations ---  
 public boolean addStudent(Student s) {  
 if (studentRegistry.containsKey(s.id)) return false;  
 studentRegistry.put(s.id, s);  
 return true;  
 }  
  
 public Student findStudent(String id) {  
 return studentRegistry.get(id);  
 }  
  
 // --- Course operations ---  
 public void addCourse(Course c) { courses.put(c.id, c); }  
  
 public void enqueueCourseRegistration(String studentId, String courseId) {  
 RegistrationRequest r = new RegistrationRequest(studentId, courseId);  
 boolean ok = registrationQueue.enqueue(r);  
 if (!ok) {  
 // fallback: use a simple LinkedList queue if circular full (unlikely in demo)  
 System.out.println("[WARN] registration queue full; rejecting request " + studentId);  
 }  
 }  
  
 public void processRegistrations() {  
 while (!registrationQueue.isEmpty()) {  
 RegistrationRequest r = registrationQueue.dequeue();  
 Course c = courses.get(r.courseId);  
 if (c == null) {  
 System.out.println("Course not found: " + r.courseId);  
 continue;  
 }  
 boolean added = c.addStudent(r.studentId);  
 if (added) {  
 System.out.println("Enrolled " + r.studentId + " to " + c.id);  
 } else {  
 System.out.println("Waitlisted " + r.studentId + " for " + c.id);  
 }  
 }  
 }  
  
 // --- Fee operations ---  
 public String recordPayment(String studentId, double amount) {  
 String txnId = "TXN" + txnCounter.incrementAndGet();  
 FeeRecord fr = new FeeRecord(txnId, studentId, amount);  
 feeRecords.put(txnId, fr);  
 // Mark fee cleared when paid in full (simple demo)  
 Student s = findStudent(studentId);  
 if (s != null) s.feeCleared = true;  
 return txnId;  
 }  
  
 public List<FeeRecord> getPaymentsForStudent(String studentId) {  
 List<FeeRecord> res = new ArrayList<>();  
 for (FeeRecord f : feeRecords.values()) {  
 if (f.studentId.equals(studentId)) res.add(f);  
 }  
 return res;  
 }  
  
 // --- Library operations ---  
 public void addBook(Book b) { books.put(b.isbn, b); }  
  
 public boolean borrowBook(String studentId, String isbn) {  
 Book b = books.get(isbn);  
 if (b == null || b.availableCopies <= 0) return false;  
 b.availableCopies -= 1;  
 borrowStack.push(isbn + ":" + studentId + ":" + java.time.LocalDate.now().toString());  
 return true;  
 }  
  
 public boolean returnBook(String studentId, String isbn) {  
 Book b = books.get(isbn);  
 if (b == null) return false;  
 b.availableCopies += 1;  
 borrowStack.push("RETURN:" + isbn + ":" + studentId + ":" + java.time.LocalDate.now().toString());  
 return true;  
 }  
  
 // --- Performance analytics ---  
 public void updateStudentGrade(String studentId, String courseId, double grade) {  
 Student s = findStudent(studentId);  
 if (s == null) return;  
 s.grades.put(courseId, grade);  
 }  
  
 public List<StudentPerformance> topKPerformers(int k) {  
 PriorityQueue<StudentPerformance> temp = new PriorityQueue<>();  
 for (Student s : studentRegistry.values()) {  
 temp.add(new StudentPerformance(s.id, s.getGPA()));  
 }  
 List<StudentPerformance> top = new ArrayList<>();  
 for (int i = 0; i < k && !temp.isEmpty(); i++) {  
 top.add(temp.poll());  
 }  
 return top;  
 }  
  
 // demo for subject correlation: set matrix  
 public void initSubjectCorrelation(int n) {  
 subjectCorrelation = new double[n][n];  
 // fill with dummy similarity values for demo  
 for (int i = 0; i < n; i++)  
 for (int j = 0; j < n; j++)  
 subjectCorrelation[i][j] = (i == j) ? 1.0 : (0.3 + (i + j) % 5 \* 0.1);  
 }  
  
 public double getSubjectCorrelation(int a, int b) {  
 if (subjectCorrelation == null) return 0.0;  
 return subjectCorrelation[a][b];  
 }  
}

## SMSDemo.java

Main demo class: sample data, runs modules, and prints summaries.

import java.util.\*;  
  
public class SMSDemo {  
 public static void main(String[] args) {  
 SchoolController controller = new SchoolController();  
  
 // --- sample students ---  
 Student s1 = new Student("S001", "Alice", "Mwangi");  
 Student s2 = new Student("S002", "Bob", "Karanja");  
 Student s3 = new Student("S003", "Cyrus", "Njoroge");  
 controller.addStudent(s1);  
 controller.addStudent(s2);  
 controller.addStudent(s3);  
  
 // --- sample courses ---  
 Course c1 = new Course("C101", "Data Structures", 2);  
 Course c2 = new Course("C102", "Database Systems", 2);  
 controller.addCourse(c1);  
 controller.addCourse(c2);  
  
 // --- sample course registrations (order matters) ---  
 controller.enqueueCourseRegistration("S001", "C101");  
 controller.enqueueCourseRegistration("S002", "C101");  
 controller.enqueueCourseRegistration("S003", "C101"); // should be waitlisted  
 controller.enqueueCourseRegistration("S003", "C102");  
 controller.processRegistrations();  
  
 // --- fee payments ---  
 System.out.println("\n[Fees]");  
 String t1 = controller.recordPayment("S001", 5000);  
 String t2 = controller.recordPayment("S002", 5000);  
 System.out.println("Recorded " + t1 + " and " + t2);  
  
 // --- library ---  
 controller.addBook(new Book("978-0131103627", "The C Programming Language", 2));  
 controller.addBook(new Book("978-0262033848", "Introduction to Algorithms", 1));  
 System.out.println("\n[Library]");  
 boolean borrowOk = controller.borrowBook("S001", "978-0131103627");  
 System.out.println("S001 borrow book1: " + borrowOk);  
 boolean borrowFail = controller.borrowBook("S002", "978-0262033848");  
 System.out.println("S002 borrow algo: " + borrowFail);  
  
 // --- performance updates ---  
 controller.updateStudentGrade("S001", "C101", 3.7);  
 controller.updateStudentGrade("S002", "C101", 3.9);  
 controller.updateStudentGrade("S003", "C102", 3.5);  
  
 // Top performers  
 System.out.println("\n[Top Performers]");  
 List<StudentPerformance> top = controller.topKPerformers(2);  
 for (StudentPerformance p : top) {  
 System.out.println(p);  
 }  
  
 // Print summary  
 System.out.println("\n[Summary]");  
 System.out.println("Students:");  
 for (Student s : controller.studentRegistry.values()) {  
 System.out.println(s);  
 }  
 System.out.println("Courses:");  
 for (Course c : controller.courses.values()) {  
 System.out.println(c + " Enrolled: " + c.enrolled);  
 }  
 System.out.println("Fee Records:");  
 for (FeeRecord fr : controller.feeRecords.values()) System.out.println(fr);  
 System.out.println("Library Books:");  
 for (Book b : controller.books.values()) System.out.println(b);  
 }  
}