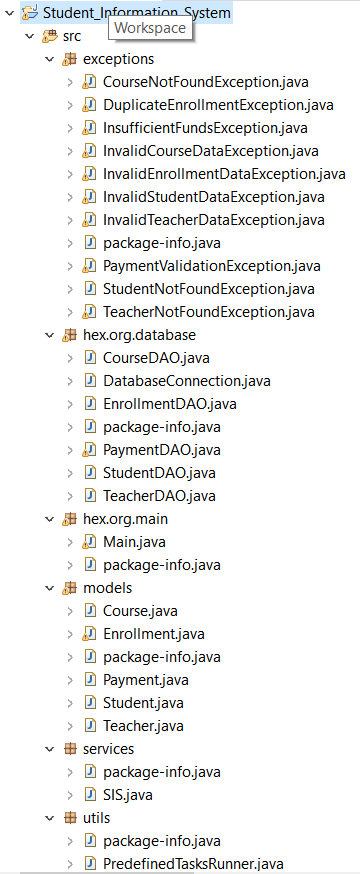
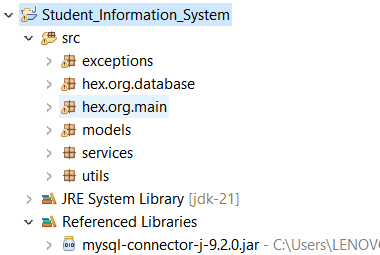
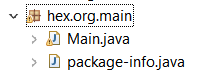
**Student Information System (SIS)**

Github link: <https://github.com/Shraddha-Gavkare/Student_Information_System.git>  
**Project Structure:**  
  
  
  
  
  
  
  
  
**Packages:**  
  
  
**Code:**  
**1) package hex.org.main**  
  
  
**Main.java**  
  
package hex.org.main;

import models.\*;

import services.SIS;

import exceptions.\*;

import java.sql.Date;

import java.sql.SQLException;

import java.time.LocalDate;

import java.util.Scanner;

public class Main {

public static void main(String[] args) throws InvalidStudentDataException, SQLException {

System.out.println("Application started\*\*\*.");

Scanner sc = new Scanner(System.in);

SIS sis = new SIS(); // SIS object created here

while (true) {

System.out.println("==========================================");

System.out.println(" Student Information System J317S ");

System.out.println("==========================================");

System.out.println("1. Add Student");

System.out.println("2. Add Teacher");

System.out.println("3. Add Course");

System.out.println("4. Enroll Student in Course");

System.out.println("5. Assign Teacher to Course");

System.out.println("6. Record Payment");

System.out.println("7. Generate Enrollment Report");

System.out.println("8. Student Enrollment(Run Task 8)");

System.out.println("9. Teacher Assignment");

System.out.println("10. Payment Record");

System.out.println("11. Enrollment Report Generation");

System.out.println("12. Generate Payment Report");

System.out.println("13. Calculate Course Statistics");

System.out.println("14. Exit");

System.out.println("==========================================");

System.out.print("Enter your choice: ");

int choice = sc.nextInt();

sc.nextLine();

try {

switch (choice) {

case 1 -> sis.addStudent(sc);

case 2 -> sis.addTeacher(sc);

case 3 -> sis.addCourse(sc);

case 4 -> sis.enrollStudent(sc);

case 5 -> sis.assignTeacherToCourse(sc);

case 6 -> sis.recordPayment(sc);

case 7 -> sis.generateEnrollmentReport(sc);

case 8 -> runTask8(sis);

case 9 -> runTask9(sis);

case 10 -> runTask10(sis);

case 11 -> runTask11(sis);

case 12 -> runTask12(sis);

case 13 -> runTask13(sis);

case 14 -> {

System.out.println("Exiting application. Goodbye!");

return;

}

default -> System.out.println("⚠️ Invalid option! Try again.");

}

} catch (Exception e) {

System.out.println("❌ Error: " + e.getMessage());

e.printStackTrace();

}

}

}

private static void runTask8(SIS sis) throws Exception {

System.out.println("\n--- Running Task 8: Student Enrollment ---");

String johnEmail = "john.doe@example.com";

Student john = sis.getStudentByEmail(johnEmail);

if (john == null) {

john = new Student(0, "John", "Doe", Date.valueOf(LocalDate.of(1995, 8, 15)), johnEmail, "1234567890");

sis.addStudent(john);

john = sis.getStudentByEmail(johnEmail);

}

Course progCourse = sis.courseDAO.getCourseByName("Introduction to Programming");

Course mathCourse = sis.courseDAO.getCourseByName("Mathematics 101");

if (john != null && progCourse != null && mathCourse != null) {

if (!sis.getEnrollmentDAO().isEnrolled(john.getId(), progCourse.getId())) {

sis.enrollStudentInCourse1(john.getId(), progCourse.getId());

System.out.println("John Doe enrolled in Introduction to Programming.");

} else {

System.out.println("John Doe is already enrolled in Introduction to Programming.");

}

if (!sis.getEnrollmentDAO().isEnrolled(john.getId(), mathCourse.getId())) {

sis.enrollStudentInCourse1(john.getId(), mathCourse.getId());

System.out.println("John Doe enrolled in Mathematics 101.");

} else {

System.out.println("John Doe is already enrolled in Mathematics 101.");

}

} else {

System.out.println("❌ Error: Could not find John Doe or the courses.");

}

}

private static void runTask9(SIS sis) throws Exception {

System.out.println("\n--- Running Task 9: Teacher Assignment ---");

String sarahEmail = "sarah.smith@example.com";

Teacher sarah = sis.getTeacherByEmail(sarahEmail);

if (sarah == null) {

sarah = new Teacher(0, "Sarah", "Smith", sarahEmail, "Computer Science");

sis.addTeacher(sarah);

sarah = sis.getTeacherByEmail(sarahEmail);

}

Course advDb = sis.courseDAO.getCourseByCode("CS302");

if (advDb == null) {

System.out.println("Inserting course with code: CS302");

advDb = new Course(0, "Advanced Database Management", "CS302", null);

sis.addCourse(advDb);

advDb = sis.courseDAO.getCourseByCode("CS302");

}

if (sarah != null && advDb != null) {

advDb.assignTeacher(sarah);

sis.updateCourse(advDb);

System.out.println("Sarah Smith assigned to Advanced Database Management.");

} else {

System.out.println("❌ Error: Could not assign Sarah Smith to the course.");

}

}

private static void runTask10(SIS sis) throws Exception {

System.out.println("\n--- Running Task 10: Payment Record ---");

String janeEmail = "jane@example.com";

Student jane = sis.getStudentByEmail(janeEmail);

if (jane == null) {

jane = new Student(0, "Jane", "Johnson", Date.valueOf("1997-03-20"), janeEmail, "9876543210");

sis.addStudent(jane);

jane = sis.getStudentByEmail(janeEmail);

}

if (jane != null) {

sis.recordPayment(jane.getEmail(), 500.00);

System.out.println("Payment recorded for Jane Johnson.");

} else {

System.out.println("❌ Error: Could not record payment for Jane Johnson.");

}

}

private static void runTask11(SIS sis) throws Exception {

System.out.println("\n--- Running Task 11: Enrollment Report Generation ---");

sis.generateEnrollmentReport("Computer Science 101");

}

private static void runTask12(SIS sis) throws Exception {

System.out.println("\n--- Running Task 12: Generate Payment Report ---");

Scanner sc = new Scanner(System.in);

System.out.print("Enter student email to generate payment report: ");

String studentEmail = sc.nextLine();

Student student = sis.getStudentByEmail(studentEmail);

if (student != null) {

sis.generatePaymentReport(student);

} else {

System.out.println("❌ Error: Student with email '" + studentEmail + "' not found.");

}

}

private static void runTask13(SIS sis) throws Exception {

System.out.println("\n--- Running Task 13: Calculate Course Statistics ---");

Scanner sc = new Scanner(System.in);

System.out.print("Enter course name to calculate statistics: ");

String courseName = sc.nextLine();

Course course = sis.courseDAO.getCourseByName(courseName);

if (course != null) {

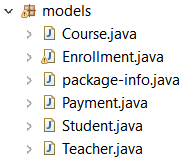
sis.calculateCourseStatistics(course);

} else {

System.out.println("❌ Error: Course with name '" + courseName + "' not found.");

}

}

}  
  
**2) package models**  
  
  
**1) Course.java**  
package models;

import java.util.ArrayList;

import java.util.List;

public class Course {

private int courseID;

private String courseName;

private String courseCode;

private Teacher instructor;

private Integer teacherId;

private List<Enrollment> enrollments;

public Course(int courseID, String courseName, String courseCode, Integer teacherId) {

this.courseID = courseID;

this.courseName = courseName;

this.courseCode = (courseCode != null) ? courseCode.trim() : null; // Trim whitespace

this.teacherId = teacherId;

this.enrollments = new ArrayList<>();

}

public Course(int courseID, String courseName, int teacherId) {

this.courseID = courseID;

this.courseName = courseName;

this.teacherId = teacherId;

this.enrollments = new ArrayList<>();

}

public void assignTeacher(Teacher teacher) {

this.instructor = teacher;

this.teacherId = teacher.getId(); // Update teacherId when assigning a Teacher object

System.out.println("Teacher assigned: " + teacher.getFirstName() + " " + teacher.getLastName());

}

public void updateCourseInfo(String courseCode, String courseName, Teacher instructor) {

this.courseCode = (courseCode != null) ? courseCode.trim() : null; // Trim whitespace

this.courseName = courseName;

this.instructor = instructor;

this.teacherId = (instructor != null) ? instructor.getId() : null; // Update teacherId

System.out.println("Course information updated successfully.");

}

public void displayCourseInfo() {

System.out.println("Course ID: " + courseID);

System.out.println("Course Name: " + courseName);

System.out.println("Course Code: " + courseCode);

System.out.println("Instructor: " + (instructor != null ? instructor.getFirstName() + " " + instructor.getLastName() : "Not Assigned"));

}

public List<Enrollment> getEnrollments() {

return enrollments;

}

public Integer getId() {

return courseID;

}

public String getCourseName() {

return courseName;

}

public String setCourseName(String courseName) {

return this.courseName = courseName;

}

public Integer getTeacherId() {

return teacherId;

}

public String getCourseCode() {

return courseCode;

}

public void setCourseCode(int i) {

this.courseCode = String.valueOf(i);

}

public Teacher getInstructor() {

return instructor;

}

public void setInstructor(Teacher instructor) {

this.instructor = instructor;

this.teacherId = (instructor != null) ? instructor.getId() : null;

}

@Override

public String toString() {

return "Course ID: " + courseID +

", Name: " + courseName +

", Code: " + courseCode +

", Instructor: " + (instructor != null ? instructor.getFirstName() + " " + instructor.getLastName() : "Not Assigned");

}

public void setCourseId(int id) {

this.courseID = id;

}

public void setTeacherId(int id) {

this.teacherId = id;

}

}  
  
**2) Enrollment.java**  
  
package models;

import java.sql.Timestamp;

import exceptions.InvalidStudentDataException;

public class Enrollment {

private static int idCounter = 1;

private int enrollmentID;

private int studentId;

private int courseId;

private Timestamp enrollmentDate;

private Student student;

private Course course;

public Enrollment(Student student, Course course) {

this.enrollmentID = idCounter++;

this.student = student;

this.course = course;

this.enrollmentDate = new Timestamp(System.currentTimeMillis());

}

public Enrollment(int enrollmentID, int studentId, int courseId, Timestamp timestamp) {

this.enrollmentID = enrollmentID;

this.student = new Student(courseId);

student.setId(studentId);

this.course = new Course(courseId, null, courseId);

course.setCourseId(courseId);

this.enrollmentDate = timestamp;

}

public Enrollment(int studentId, int courseId) {

this.studentId = studentId;

this.courseId = courseId;

}

public int getStudentId() {

return studentId;

}

public int getCourseId() {

return courseId;

}

// Getters

public int getEnrollmentID() {

return enrollmentID;

}

public Timestamp getEnrollmentDate() {

return enrollmentDate;

}

public Course getCourse() {

return course;

}

public Student getStudent() {

return student;

}

public int setEnrollmentId(int enrollmentID) {

return enrollmentID;

}

public int setCourseId(int i) {

return i;

}

public Course setcourse() {

return course;

}

public Student setStudent(Student student) {

return student;

}

}  
  
**3) Payment.java**  
  
package models;

import java.math.BigDecimal;

import java.time.LocalDateTime;

public class Payment {

private static int idCounter = 1;

private int paymentID;

private int studentId;

private BigDecimal amount;

private LocalDateTime paymentDate;

// Constructor for inserting new payments (without manual paymentID)

public Payment(int studentId, double amount, LocalDateTime paymentDate) {

this.paymentID = idCounter++;

this.studentId = studentId;

this.amount = BigDecimal.valueOf(amount);

this.paymentDate = paymentDate;

}

// Constructor for retrieving from database

public Payment(int paymentID, int studentId, BigDecimal amount, LocalDateTime paymentDate) {

this.paymentID = paymentID;

this.studentId = studentId;

this.amount = amount;

this.paymentDate = paymentDate;

}

public int getPaymentID() {

return paymentID;

}

public int getStudentId() {

return studentId;

}

public BigDecimal getPaymentAmount() {

return amount;

}

public LocalDateTime getPaymentDate() {

return paymentDate;

}

public void displayPaymentInfo() {

System.out.println("Payment ID: " + paymentID);

System.out.println("Student ID: " + studentId);

System.out.println("Amount: " + amount);

System.out.println("Date: " + paymentDate);

}

}

1. **Student.java**  
     
   package models;

import exceptions.InvalidStudentDataException;

import java.util.ArrayList;

import java.util.List;

import java.sql.Date;

import java.time.LocalDate;

import java.time.LocalDateTime;

public class Student {

private int studentId;

private String firstName;

private String lastName;

private Date dateOfBirth;

private String email;

private String phoneNumber;

private List<Enrollment> enrollments;

private List<Payment> payments;

public Student() {

this.enrollments = new ArrayList<>();

this.payments = new ArrayList<>();

}

public Student(int studentId) {

this.studentId = studentId;

this.enrollments = new ArrayList<>();

this.payments = new ArrayList<>();

}

public Student(int studentId, String firstName, String lastName, Date dateOfBirth, String email, String phoneNumber) throws InvalidStudentDataException {

System.out.println("Creating student with email: " + email); // 🪵 Log email

validateStudentData(email, phoneNumber);

this.studentId = studentId;

this.firstName = firstName;

this.lastName = lastName;

this.dateOfBirth = dateOfBirth;

this.email = email;

this.phoneNumber = phoneNumber;

this.enrollments = new ArrayList<>();

this.payments = new ArrayList<>();

}

public Student(String firstName, String lastName, String dob, String email, String phoneNumber) throws InvalidStudentDataException {

System.out.println("Creating student:");

System.out.println(" Name: " + firstName + " " + lastName);

System.out.println(" DOB: " + dob);

System.out.println(" Email: " + email); // 🪵 Log email

System.out.println(" Phone: " + phoneNumber);

validateStudentData(email, phoneNumber);

this.firstName = firstName;

this.lastName = lastName;

this.dateOfBirth = Date.valueOf(dob); // yyyy-MM-dd

this.email = email;

this.phoneNumber = phoneNumber;

this.enrollments = new ArrayList<>();

this.payments = new ArrayList<>();

}

private void validateStudentData(String email, String phoneNumber) throws InvalidStudentDataException {

if (email == null || email.isEmpty()) {

throw new InvalidStudentDataException("Email cannot be empty.");

}

if (phoneNumber == null || phoneNumber.length() != 10) {

throw new InvalidStudentDataException("Phone number must be 10 digits.");

}

}

public void enrollInCourse(Course course) {

for (Enrollment e : enrollments) {

if (e.getCourse().equals(course)) {

System.out.println("Already enrolled in course: " + course.getCourseName());

return;

}

}

Enrollment enrollment = new Enrollment(this, course);

enrollments.add(enrollment);

System.out.println(firstName + " has been enrolled in " + course.getCourseName());

}

public void updateStudentInfo(String firstName, String lastName, Date dateOfBirth, String email, String phoneNumber) throws InvalidStudentDataException {

validateStudentData(email, phoneNumber);

this.firstName = firstName;

this.lastName = lastName;

this.dateOfBirth = dateOfBirth;

this.email = email;

this.phoneNumber = phoneNumber;

System.out.println("Student information updated successfully.");

}

public void makePayment(double amount, LocalDateTime paymentDate) throws InvalidStudentDataException {

if (amount <= 0) {

throw new InvalidStudentDataException("Payment amount must be greater than zero.");

}

// Create Payment using only student ID

Payment payment = new Payment(this.getId(), amount, paymentDate);

payments.add(payment); // Assuming 'payments' is your List<Payment> or similar

System.out.println("Payment of " + amount + " recorded on " + paymentDate);

}

public void displayStudentInfo() {

System.out.println("Student ID: " + studentId);

System.out.println("Name: " + firstName + " " + lastName);

System.out.println("Date of Birth: " + dateOfBirth);

System.out.println("Email: " + email);

System.out.println("Phone: " + phoneNumber);

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public Date getDateOfBirth() {

return dateOfBirth;

}

public void setDateOfBirth(LocalDate localDate) {

this.dateOfBirth = Date.valueOf(localDate);

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getPhoneNumber() {

return phoneNumber;

}

public void setPhoneNumber(String phoneNumber) {

this.phoneNumber = phoneNumber;

}

public void setId(int student) {

this.studentId = student;

}

public Integer getId() {

return studentId;

}

public List<Enrollment> getEnrollments() {

return enrollments;

}

public void setEnrollments(List<Enrollment> enrollments) {

this.enrollments = enrollments;

}

public void addPayment(Payment payment) {

this.payments.add(payment);

}

public List<Payment> getPaymentHistory() {

return payments;

}

@Override

public String toString() {

return "Student ID: " + studentId +

", Name: " + firstName + " " + lastName +

", Email: " + email +

", Phone: " + phoneNumber;

}

public void setAddress(String address) {

System.out.println("Address set to: " + address);

}

}

5) **Teacher.java**  
package models;

import java.util.ArrayList;

import java.util.List;

public class Teacher {

private int teacherID;

private String firstName;

private String lastName;

private String email;

private String expertise;

private List<Course> assignedCourses;

// Constructor for new teachers (without ID)

public Teacher(String firstName, String lastName, String email, String expertise) {

this.firstName = firstName;

this.lastName = lastName;

this.email = email;

this.expertise = expertise;

this.assignedCourses = new ArrayList<>();

}

// Constructor for database records (with ID)

public Teacher(int teacherID, String firstName, String lastName, String email, String expertise) {

this.teacherID = teacherID;

this.firstName = firstName;

this.lastName = lastName;

this.email = email;

this.expertise = expertise;

this.assignedCourses = new ArrayList<>();

}

public int getTeacherID() {

return teacherID;

}

public void setTeacherID(int teacherID) {

this.teacherID = teacherID;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getExpertise() {

return expertise;

}

public void setExpertise(String expertise) {

this.expertise = expertise;

}

public List<Course> getAssignedCourses() {

return assignedCourses;

}

public void assignCourse(Course course) {

assignedCourses.add(course);

System.out.println("Course " + course.getCourseName() + " assigned to " + firstName + " " + lastName);

}

public void displayTeacherInfo() {

System.out.println("Teacher ID: " + teacherID);

System.out.println("Name: " + firstName + " " + lastName);

System.out.println("Email: " + email);

System.out.println("Expertise: " + expertise);

}

public int getId() {

return teacherID;

}

@Override

public String toString() {

return "Teacher ID: " + teacherID +

", Name: " + firstName + " " + lastName +

", Email: " + email +

", Expertise: " + expertise;

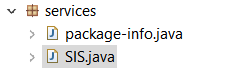
}

public void setId(int id) {

this.teacherID = id;

}

}  
  
**3) package services**

  
  
**SIS.java**  
  
package services;

import models.\*;

import exceptions.\*;

import hex.org.database.\*;

import java.sql.Date;

import java.sql.SQLException;

import java.time.LocalDateTime;

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class SIS {

private final StudentDAO studentDAO;

private final TeacherDAO teacherDAO;

public final CourseDAO courseDAO = new CourseDAO();

private final EnrollmentDAO enrollmentDAO = new EnrollmentDAO();

private final PaymentDAO paymentDAO = new PaymentDAO();

private List<Student> students;

private List<Course> courses;

private List<Teacher> teachers;

public SIS() throws SQLException {

this.students = new ArrayList<>();

this.courses = new ArrayList<>();

this.teachers = new ArrayList<>();

this.studentDAO = new StudentDAO();

this.teacherDAO = new TeacherDAO();

}

public EnrollmentDAO getEnrollmentDAO() {

return enrollmentDAO;

}

// STUDENT OPERATIONS

public void addStudent(Student student) throws InvalidStudentDataException {

if (student.getEmail() == null || student.getEmail().isEmpty()) {

throw new InvalidStudentDataException("Email cannot be empty.");

}

studentDAO.insertStudent(student);

students.add(student);

System.out.println("Student added successfully.");

}

public void updateStudent(Student student) {

studentDAO.updateStudent(student);

System.out.println("Student updated successfully.");

}

// TEACHER OPERATIONS

public void addTeacher(Teacher teacher) throws InvalidTeacherDataException {

if (teacher.getEmail() == null || teacher.getEmail().isEmpty()) {

throw new InvalidTeacherDataException("Email cannot be empty.");

}

teacherDAO.insertTeacher(teacher);

teachers.add(teacher);

System.out.println("Teacher added successfully.");

}

public void updateTeacher(Teacher teacher) {

teacherDAO.updateTeacher(teacher);

System.out.println("Teacher updated successfully.");

}

// COURSE OPERATIONS

public void addCourse(Course course) throws InvalidCourseDataException {

if (course.getCourseName() == null || course.getCourseName().isEmpty()) {

throw new InvalidCourseDataException("Course name cannot be empty.");

}

courseDAO.insertCourse(course);

courses.add(course);

System.out.println("Course added successfully.");

}

public void updateCourse(Course course) {

courseDAO.updateCourse(course);

System.out.println("Course updated successfully.");

}

public void enrollStudent(Scanner sc) {

try {

System.out.print("Enter student email: ");

String email = sc.nextLine();

Student student = studentDAO.getStudentByEmail(email);

if (student == null) {

System.out.println("Error: Student not found.");

return;

}

Integer studentId = student.getId(); // Get the Integer ID

if (studentId == null) {

System.out.println("Error: Could not retrieve student ID.");

return;

}

System.out.print("Enter course name: ");

String courseName = sc.nextLine();

Course course = courseDAO.getCourseByName(courseName);

if (course == null) {

System.out.println("Error: Course not found.");

return;

}

Integer courseId = course.getId(); // Get the Integer ID

if (courseId == null) {

System.out.println("Error: Could not retrieve course ID.");

return;

}

enrollStudentInCourse1(studentId, courseId);

System.out.println("\*\*\*Student enrolled successfully!"); // Add success message

} catch (Exception e) {

System.out.println("Error enrolling student.");

e.printStackTrace();

}

}

// PAYMENT

public void recordPayment(String email, double amount)

throws PaymentValidationException, StudentNotFoundException, InvalidStudentDataException {

Student student = studentDAO.getStudentByEmail(email);

if (student == null)

throw new StudentNotFoundException("Student not found.");

if (amount <= 0)

throw new PaymentValidationException("Amount must be greater than zero.");

LocalDateTime dateTime = LocalDateTime.now();

Payment payment = new Payment(student.getId(), amount, dateTime);

paymentDAO.insertPayment(payment);

student.getPaymentHistory().add(payment); // Optional: You can use a helper method for display

System.out.println("Payment recorded successfully.");

}

// Overloaded method (optional/future use)

public void recordPayment(Student student, double amount, Date date) {

if (student != null && amount > 0 && date != null) {

LocalDateTime dateTime = date.toLocalDate().atStartOfDay();

Payment payment = new Payment(student.getId(), amount, dateTime);

paymentDAO.insertPayment(payment);

student.getPaymentHistory().add(payment);

System.out.println("Payment recorded successfully (with custom date).");

}

}

// DISPLAY

public void displayAllStudents() throws InvalidStudentDataException {

List<Student> studentsFromDB = studentDAO.getAllStudents();

for (Student student : studentsFromDB) {

System.out.println("Student Name: " + student.getFirstName() + " " + student.getLastName());

}

}

public void generateEnrollmentReport(String courseName) {

Course course = courseDAO.getCourseByName(courseName);

if (course == null) {

System.out.println("Course not found: " + courseName);

return;

}

int courseId = course.getId();

List<Student> enrolledStudents = enrollmentDAO.getEnrolledStudentsByCourseId(courseId);

System.out.println("\n Enrollment Report for: " + course.getCourseName());

for (Student s : enrolledStudents) {

System.out.println(" - " + s.getFirstName() + " " + s.getLastName());

}

}

// UTILITY METHOD

public Student getStudentByEmail(String email) throws InvalidStudentDataException {

return studentDAO.getStudentByEmail(email);

}

// UI USING SCANNER

public void addStudent(Scanner sc) {

try {

System.out.print("First Name: ");

String first = sc.nextLine();

System.out.print("Last Name: ");

String last = sc.nextLine();

System.out.print("DOB (yyyy-mm-dd): ");

String dobStr = sc.nextLine();

System.out.print("Email: ");

String email = sc.nextLine();

System.out.print("Phone: ");

String phone = sc.nextLine();

Student student = new Student(0, first, last, Date.valueOf(dobStr), email, phone);

addStudent(student);

} catch (Exception e) {

System.out.println("Error adding student: " + e.getMessage());

}

}

public void addTeacher(Scanner sc) {

try {

System.out.print("First Name: ");

String first = sc.nextLine();

System.out.print("Last Name: ");

String last = sc.nextLine();

System.out.print("Email: ");

String email = sc.nextLine();

System.out.print("Department: ");

String dept = sc.nextLine();

Teacher teacher = new Teacher(0, first, last, email, dept);

addTeacher(teacher);

} catch (Exception e) {

System.out.println("Error adding teacher: " + e.getMessage());

}

}

public void addCourse(Scanner sc) {

try {

System.out.print("Course Name: ");

String name = sc.nextLine();

System.out.print("Course Code: ");

String code = sc.nextLine();

Course course = new Course(0, name, code, null);

addCourse(course);

} catch (Exception e) {

System.out.println("Error adding course: " + e.getMessage());

}

}

public void enrollStudentInCourse1(Integer studentId, Integer courseId) {

if (studentId == null || courseId == null) {

System.out.println("Cannot enroll. Student ID or Course ID is null.");

return;

}

enrollmentDAO.enrollStudent(studentId, courseId);

}

public void assignTeacherToCourse(Scanner sc) {

try {

System.out.print("Enter teacher email: ");

String email = sc.nextLine();

Teacher teacher = teacherDAO.getTeacherByEmail(email);

if (teacher == null) throw new TeacherNotFoundException("Teacher not found.");

System.out.print("Enter course name: ");

String courseName = sc.nextLine();

Course course = courseDAO.getCourseByName(courseName);

if (course == null) throw new CourseNotFoundException("Course not found.");

course.setTeacherId(teacher.getId());

updateCourse(course);

} catch (Exception e) {

System.out.println("Error assigning teacher: " + e.getMessage());

}

}

public void recordPayment(Scanner sc) {

try {

System.out.print("Enter student email: ");

String email = sc.nextLine();

System.out.print("Enter amount: ");

double amount = Double.parseDouble(sc.nextLine());

recordPayment(email, amount);

} catch (Exception e) {

System.out.println("Error recording payment: " + e.getMessage());

}

}

public void generateEnrollmentReport(Scanner sc) {

System.out.print("Enter course name: ");

String courseName = sc.nextLine();

generateEnrollmentReport(courseName);

}

public Teacher getTeacherByEmail(String email) throws SQLException {

// First, check the in-memory list

for (Teacher teacher : teachers) {

if (teacher.getEmail().equalsIgnoreCase(email)) {

return teacher;

}

}

// If not found in memory, fetch from the database

return teacherDAO.getTeacherByEmail(email);

}

public EnrollmentDAO getEnrollmentDAO1() {

return enrollmentDAO;

}

// Generate Payment Report for a specific student

public void generatePaymentReport(Student student) throws InvalidStudentDataException {

System.out.println("\n--- Payment Report for Student: " + student.getFirstName() + " " + student.getLastName() + " ---");

List<Payment> paymentHistory = paymentDAO.getPaymentsByStudentId(student.getId());

if (paymentHistory.isEmpty()) {

System.out.println("No payment history found for this student.");

} else {

for (Payment payment : paymentHistory) {

System.out.println("Date: " + payment.getPaymentDate() + ", Amount: $" + payment.getPaymentAmount());

}

double totalPaid = paymentHistory.stream()

.mapToDouble(payment -> payment.getPaymentAmount().doubleValue())

.sum();

System.out.println("--------------------------------------------------");

System.out.println("Total Amount Paid: $" + totalPaid);

}

}

// Calculate statistics for a specific course

public void calculateCourseStatistics(Course course) {

System.out.println("\n--- Course Statistics for: " + course.getCourseName() + " (" + course.getCourseCode() + ") ---");

// Number of Enrollments

List<Student> enrolledStudents = enrollmentDAO.getEnrolledStudentsByCourseId(course.getId());

int numberOfEnrollments = enrolledStudents.size();

System.out.println("Number of Enrollments: " + numberOfEnrollments);

// Total Payments for the course (assuming each enrolled student paid a standard fee)

// This is a simplified calculation. A more complex system might track individual payment per enrollment.

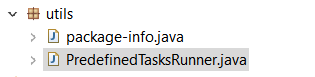
double standardCourseFee = 500.0; // Example standard fee

double totalPayments = numberOfEnrollments \* standardCourseFee;

System.out.println("Estimated Total Payments (at $" + standardCourseFee + " per student): $" + totalPayments);

// You can add more statistics here, such as average payment, etc.

}

}  
  
**4)package utils**  
  
  
**PredefinedTasksRunner.java**  
  
package utils;

import hex.org.database.StudentDAO;

import exceptions.InvalidStudentDataException;

import java.sql.SQLException;

import hex.org.database.CourseDAO;

import hex.org.database.EnrollmentDAO;

import models.Student;

import models.Course;

import models.Enrollment;

public class PredefinedTasksRunner {

public static void runPredefinedTasks() throws InvalidStudentDataException, SQLException {

StudentDAO studentDAO = new StudentDAO();

CourseDAO courseDAO = new CourseDAO();

EnrollmentDAO enrollmentDAO = new EnrollmentDAO();

// Student data

String firstName = "John";

String lastName = "Doe";

String dob = "2000-05-15";

String email = "john.doe@example.com";

String phone = "9876543210";

// Check or create student

Student john = studentDAO.getStudentByEmail(email);

if (john == null) {

System.out.println("Creating student with email: " + email);

john = new Student(firstName, lastName, dob, email, phone); // ✅ Correct constructor

studentDAO.insertStudent(john);

}

// Check or create course

Course course1 = courseDAO.getCourseByName("Introduction to Programming");

if (course1 == null) {

course1 = new Course(0, "Introduction to Programming", "CS101", null);

courseDAO.insertCourse(course1);

}

// Enroll if not already

if (!enrollmentDAO.isEnrolled(john.getId(), course1.getId())) {

Enrollment enrollment = new Enrollment(john.getId(), course1.getId());

enrollmentDAO.insertEnrollment(enrollment);

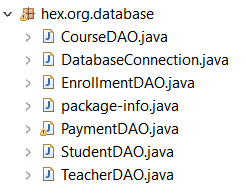
System.out.println("Enrolled John in Introduction to Programming.");

} else {

System.out.println("John is already enrolled in Introduction to Programming.");

}

}

}  
  
**5) package hex.org.database**  


1. **CourseDAO.java**  
     
   package hex.org.database;

import models.Course;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

public class CourseDAO {

public List<Course> getAllCourses() {

List<Course> courses = new ArrayList<>();

String query = "SELECT \* FROM courses";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query);

ResultSet rs = stmt.executeQuery()) {

while (rs.next()) {

Course course = new Course(

rs.getInt("id"),

rs.getString("course\_name"),

rs.getObject("teacher\_id") != null ? rs.getInt("teacher\_id") : null

);

courses.add(course);

}

} catch (SQLException e) {

e.printStackTrace();

}

return courses;

}

private boolean teacherExists(Integer teacherId) {

if (teacherId == null) return true;

String query = "SELECT COUNT(\*) FROM teachers WHERE id = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, teacherId);

ResultSet rs = stmt.executeQuery();

if (rs.next()) return rs.getInt(1) > 0;

} catch (SQLException e) {

e.printStackTrace();

}

return false;

}

public void insertCourse(Course course) {

if (!teacherExists(course.getTeacherId())) {

System.out.println("Cannot insert course: Teacher ID " + course.getTeacherId() + " does not exist.");

return;

}

String query = "INSERT INTO courses (course\_name, teacher\_id, code) VALUES (?, ?, ?)"; // Added 'code' to the insert columns

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query, Statement.RETURN\_GENERATED\_KEYS)) {

stmt.setString(1, course.getCourseName());

stmt.setObject(2, course.getTeacherId());

stmt.setString(3, course.getCourseCode()); // Set the 'code' here

int affectedRows = stmt.executeUpdate();

if (affectedRows > 0) {

try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {

if (generatedKeys.next()) {

course.setCourseId(generatedKeys.getInt(1)); // Use setCourseId for the primary key 'id'

}

}

System.out.println("Course inserted successfully with ID: " + course.getId()); // Use getId()

}

} catch (SQLException e) {

e.printStackTrace();

}

}

public void updateCourse(Course course) {

String query = "UPDATE courses SET course\_name = ?, teacher\_id = ? WHERE id = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setString(1, course.getCourseName());

stmt.setObject(2, course.getTeacherId());

stmt.setInt(3, course.getId());

int rowsUpdated = stmt.executeUpdate();

if (rowsUpdated > 0) {

System.out.println("Course updated successfully.");

} else {

System.out.println("Course not found.");

}

} catch (SQLException e) {

e.printStackTrace();

}

}

public Course getCourseByName(String integer2) {

Course course = null;

String query = "SELECT \* FROM courses WHERE course\_name = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setNString(1, integer2);

ResultSet rs = stmt.executeQuery();

if (rs.next()) {

int id = rs.getInt("id");

String code = rs.getString("code");

course = new Course(id, integer2, code, null);

}

} catch (SQLException e) {

e.printStackTrace();

}

if (course == null) {

System.out.println("Course not found for name: " + integer2);

} else {

System.out.println("Found course: " + course.getId() + ", " + course.getCourseName());

}

return course;

}

public Course getCourseById(int courseId) {

String query = "SELECT \* FROM courses WHERE id = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, courseId);

ResultSet rs = stmt.executeQuery();

if (rs.next()) {

return new Course(

rs.getInt("id"),

rs.getString("course\_name"),

rs.getObject("teacher\_id") != null ? rs.getInt("teacher\_id") : null

);

}

} catch (SQLException e) {

e.printStackTrace();

}

return null;

}

public Course getCourseByCode(String courseCode) {

String query = "SELECT id, course\_name, code, teacher\_id FROM courses WHERE code = ?"; // Use 'code' in SELECT and WHERE

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setString(1, courseCode);

ResultSet rs = stmt.executeQuery();

if (rs.next()) {

return new Course(

rs.getInt("id"),

rs.getString("course\_name"),

rs.getString("code"),

rs.getObject("teacher\_id") == null ? null : rs.getInt("teacher\_id")

);

}

} catch (SQLException e) {

e.printStackTrace();

}

return null;

}

}

2) **DatabaseConnection.java**  
package hex.org.database;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class DatabaseConnection {

private static final String URL = "jdbc:mysql://localhost:3306/sis\_db";

private static final String USER = "root";

private static final String PASSWORD = "Shraddha@2003";

public static Connection getConnection() throws SQLException {

Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);

System.out.println("Connected to Database successfully!!!!");

return conn;

}

}

**3) EnrollmentDAO.java**  
  
package hex.org.database;

import models.Course;

import models.Enrollment;

import models.Student;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

import exceptions.InvalidStudentDataException;

public class EnrollmentDAO {

public void enrollStudent(int studentId, int courseId) {

String query = "INSERT INTO enrollments (student\_id, course\_id, enrolled\_at) VALUES (?, ?, ?)";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, studentId);

stmt.setInt(2, courseId);

stmt.setTimestamp(3, new Timestamp(System.currentTimeMillis()));

stmt.executeUpdate();

System.out.println("Student enrollment successfull\*\*.");

} catch (SQLException e) {

e.printStackTrace();

}

}

public List<Enrollment> getEnrollmentsForStudent(int studentId) throws InvalidStudentDataException {

List<Enrollment> enrollments = new ArrayList<>();

String query = "SELECT \* FROM enrollments WHERE student\_id = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, studentId);

ResultSet rs = stmt.executeQuery();

while (rs.next()) {

Enrollment enrollment = new Enrollment(

rs.getInt("id"),

rs.getInt("student\_id"),

rs.getInt("course\_id"),

rs.getTimestamp("enrolled\_at")

);

enrollments.add(enrollment);

}

} catch (SQLException e) {

e.printStackTrace();

}

return enrollments;

}

// Insert a new enrollment using Enrollment object

public void insertEnrollment(Enrollment enrollment) {

String query = "INSERT INTO enrollments (student\_id, course\_id, enrolled\_at) VALUES (?, ?, ?)";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, enrollment.getStudentId());

stmt.setInt(2, enrollment.getCourseId());

stmt.setTimestamp(3, new Timestamp(System.currentTimeMillis()));

stmt.executeUpdate();

System.out.println("Enrollment added successfully.");

} catch (SQLException e) {

e.printStackTrace();

}

}

// Update existing enrollment

public void updateEnrollment(Enrollment enrollment) {

String query = "UPDATE enrollments SET course\_id = ?, enrolled\_at = ? WHERE id = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, enrollment.getCourseId());

stmt.setTimestamp(2, new Timestamp(System.currentTimeMillis()));

stmt.setInt(3, enrollment.getEnrollmentID());

int rowsUpdated = stmt.executeUpdate();

if (rowsUpdated > 0) {

System.out.println("Enrollment updated successfully.");

} else {

System.out.println("⚠️ Enrollment not found.");

}

} catch (SQLException e) {

e.printStackTrace();

}

}

public void insertEnrollment(Integer studentId, Integer courseId) {

String query = "INSERT INTO enrollments (student\_id, course\_id, enrolled\_at) VALUES (?, ?, ?)";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, studentId);

stmt.setInt(2, courseId);

stmt.setTimestamp(3, new Timestamp(System.currentTimeMillis()));

stmt.executeUpdate();

System.out.println("Enrollment added using Student and Course IDs.");

} catch (SQLException e) {

e.printStackTrace();

}

}

public List<Student> getEnrolledStudentsByCourseId(int courseId) {

List<Student> students = new ArrayList<>();

String query = "SELECT s.id, s.first\_name, s.last\_name, s.email, s.phone\_number, s.date\_of\_birth " +

"FROM students s " +

"JOIN enrollments e ON s.id = e.student\_id " +

"WHERE e.course\_id = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, courseId);

ResultSet rs = stmt.executeQuery();

while (rs.next()) {

Student student = new Student();

student.setId(rs.getInt("id"));

student.setFirstName(rs.getString("first\_name"));

student.setLastName(rs.getString("last\_name"));

student.setEmail(rs.getString("email"));

student.setPhoneNumber(rs.getString("phone\_number"));

student.setDateOfBirth(rs.getDate("date\_of\_birth").toLocalDate());

students.add(student);

}

} catch (SQLException e) {

e.printStackTrace();

}

return students;

}

// Check if a student is already enrolled in a course

public boolean isEnrolled(int studentId, int courseId) {

String query = "SELECT COUNT(\*) FROM enrollments WHERE student\_id = ? AND course\_id = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, studentId);

stmt.setInt(2, courseId);

ResultSet rs = stmt.executeQuery();

if (rs.next()) {

return rs.getInt(1) > 0;

}

} catch (SQLException e) {

e.printStackTrace();

}

return false;

}

public List<Enrollment> getEnrollmentsByStudentId(Integer studentId) {

List<Enrollment> enrollments = new ArrayList<>();

String query = "SELECT e.id AS enrollment\_id, e.course\_id, e.enrolled\_at, c.course\_name " +

"FROM enrollments e " +

"JOIN courses c ON e.course\_id = c.id " +

"WHERE e.student\_id = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, studentId);

ResultSet rs = stmt.executeQuery();

while (rs.next()) {

Enrollment enrollment = new Enrollment(studentId, studentId);

enrollment.setEnrollmentId(studentId);

enrollment.setCourseId(rs.getInt("course\_id"));

enrollment.setEnrollmentId(rs.getInt("enrollment\_id"));

Course course = new Course(studentId, query, studentId);

course.setCourseId(rs.getInt("course\_id"));

course.setCourseName(rs.getString("course\_name"));

enrollment.setCourseId(rs.getInt("course\_id"));

Student student = new Student();

student.setId(studentId);

enrollment.setStudent(student);

enrollments.add(enrollment);

}

} catch (SQLException e) {

e.printStackTrace();

}

return enrollments;

}

}

**4) PaymentDAO.java**  
package hex.org.database;

import models.Payment;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

import java.time.LocalDateTime;

public class PaymentDAO {

// Retrieve all payments for a specific student

public List<Payment> getPaymentsForStudent(int studentId) {

List<Payment> payments = new ArrayList<>();

String query = "SELECT \* FROM payments WHERE student\_id = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, studentId);

ResultSet rs = stmt.executeQuery();

while (rs.next()) {

Payment payment = new Payment(

rs.getInt("id"),

rs.getInt("student\_id"),

rs.getBigDecimal("amount"),

rs.getTimestamp("payment\_date").toLocalDateTime()

);

payments.add(payment);

}

} catch (SQLException e) {

e.printStackTrace();

}

return payments;

}

// Insert a new payment record

public void insertPayment(Payment payment) {

String query = "INSERT INTO payments (student\_id, amount, payment\_date) VALUES (?, ?, ?)";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, payment.getStudentId());

stmt.setBigDecimal(2, payment.getPaymentAmount());

stmt.setTimestamp(3, Timestamp.valueOf(payment.getPaymentDate()));

stmt.executeUpdate();

System.out.println("Payment inserted successfully.");

} catch (SQLException e) {

e.printStackTrace();

}

}

// Update an existing payment record

public void updatePayment(Payment payment) {

String query = "UPDATE payments SET amount = ?, payment\_date = ? WHERE id = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setBigDecimal(1, payment.getPaymentAmount());

stmt.setTimestamp(2, Timestamp.valueOf(payment.getPaymentDate()));

stmt.setInt(3, payment.getPaymentID());

int rowsUpdated = stmt.executeUpdate();

if (rowsUpdated > 0) {

System.out.println("Payment updated successfully.");

} else {

System.out.println("Payment not found.");

}

} catch (SQLException e) {

e.printStackTrace();

}

}

// Retrieve payments by student ID (alternative method)

public List<Payment> getPaymentsByStudentId(Integer studentId) {

List<Payment> payments = new ArrayList<>();

String query = "SELECT id, student\_id, amount, payment\_date FROM payments WHERE student\_id = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, studentId);

ResultSet rs = stmt.executeQuery();

while (rs.next()) {

Payment payment = new Payment(

rs.getInt("id"),

rs.getInt("student\_id"),

rs.getBigDecimal("amount"),

rs.getTimestamp("payment\_date").toLocalDateTime()

);

payments.add(payment);

}

} catch (SQLException e) {

e.printStackTrace();

}

return payments;

}

}

**5) StudentDAO.java**  
package hex.org.database;

import models.Student;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

import exceptions.InvalidStudentDataException;

public class StudentDAO {

private Connection conn;

public StudentDAO() throws SQLException {

conn = DatabaseConnection.getConnection(); // Your DB connection logic

}

public Student getStudentByEmail1(String email) throws InvalidStudentDataException {

try {

String sql = "SELECT \* FROM students WHERE email = ?";

PreparedStatement ps = conn.prepareStatement(sql);

ps.setString(1, email);

ResultSet rs = ps.executeQuery();

if (rs.next()) {

return new Student(

rs.getInt("id"),

rs.getString("first\_name"),

rs.getString("last\_name"),

rs.getDate("date\_of\_birth"),

rs.getString("email"),

rs.getString("phone\_number")

);

}

} catch (SQLException e) {

e.printStackTrace();

}

return null;

}

public List<Student> getAllStudents() throws InvalidStudentDataException {

List<Student> students = new ArrayList<>();

String query = "SELECT \* FROM students";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query);

ResultSet rs = stmt.executeQuery()) {

while (rs.next()) {

Student student = new Student(

rs.getInt("id"),

rs.getString("first\_name"),

rs.getString("last\_name"),

rs.getDate("date\_of\_birth"),

rs.getString("email"),

rs.getString("phone\_number")

);

students.add(student);

}

} catch (SQLException e) {

e.printStackTrace();

}

return students;

}

public Student getStudentById(int studentId) throws InvalidStudentDataException {

String query = "SELECT \* FROM students WHERE id = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, studentId); // Pass the int directly

ResultSet rs = stmt.executeQuery();

if (rs.next()) {

return new Student(

rs.getInt("id"),

rs.getString("first\_name"),

rs.getString("last\_name"),

rs.getDate("date\_of\_birth"),

rs.getString("email"),

rs.getString("phone\_number")

);

}

} catch (SQLException e) {

e.printStackTrace();

}

return null;

}

public boolean studentExists(String email) {

String query = "SELECT COUNT(\*) FROM students WHERE email = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setString(1, email);

ResultSet rs = stmt.executeQuery();

if (rs.next()) return rs.getInt(1) > 0;

} catch (SQLException e) {

e.printStackTrace();

}

return false;

}

public void insertStudent(Student student) throws InvalidStudentDataException {

if (student.getEmail() == null || student.getEmail().trim().isEmpty()) {

throw new InvalidStudentDataException("Email cannot be empty.");

}

if (studentExists(student.getEmail())) {

System.out.println("Student with this email already exists##.");

return;

}

String query = "INSERT INTO students (first\_name, last\_name, email, date\_of\_birth, phone\_number) VALUES (?, ?, ?, ?, ?)";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query, Statement.RETURN\_GENERATED\_KEYS)) {

stmt.setString(1, student.getFirstName());

stmt.setString(2, student.getLastName());

stmt.setString(3, student.getEmail());

stmt.setDate(4, student.getDateOfBirth());

stmt.setString(5, student.getPhoneNumber());

int affectedRows = stmt.executeUpdate();

if (affectedRows > 0) {

try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {

if (generatedKeys.next()) {

student.setId(generatedKeys.getInt(1));

}

}

System.out.println("Student inserted successfully with ID: " + student.getId());

}

} catch (SQLException e) {

e.printStackTrace();

}

}

public void updateStudent(Student student) {

String query = "UPDATE students SET first\_name = ?, last\_name = ?, email = ?, date\_of\_birth = ?, phone\_number = ? WHERE id = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setString(1, student.getFirstName());

stmt.setString(2, student.getLastName());

stmt.setString(3, student.getEmail());

stmt.setDate(4, student.getDateOfBirth());

stmt.setString(5, student.getPhoneNumber());

stmt.setInt(6, student.getId());

int rowsUpdated = stmt.executeUpdate();

if (rowsUpdated > 0) {

System.out.println("Student updated successfully.");

} else {

System.out.println("⚠Student not found.");

}

} catch (SQLException e) {

e.printStackTrace();

}

}

public Student getStudentByEmail(String email) throws InvalidStudentDataException {

String query = "SELECT \* FROM students WHERE email = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setNString(1, email);

ResultSet rs = stmt.executeQuery();

if (rs.next()) {

return new Student(

rs.getInt("id"),

rs.getString("first\_name"),

rs.getString("last\_name"),

rs.getDate("date\_of\_birth"),

rs.getString("email"),

rs.getString("phone\_number")

);

}

} catch (SQLException e) {

e.printStackTrace();

}

return null;

}

}  
  
**6) TeacherDAO.java**  
package hex.org.database;

import models.Teacher;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

public class TeacherDAO {

private Connection conn;

public TeacherDAO() throws SQLException {

conn = DatabaseConnection.getConnection(); // Your DB connection logic

}

public Teacher getTeacherByEmail1(String email) {

try {

String sql = "SELECT \* FROM teachers WHERE email = ?";

PreparedStatement ps = conn.prepareStatement(sql);

ps.setString(1, email);

ResultSet rs = ps.executeQuery();

if (rs.next()) {

return new Teacher(

rs.getInt("id"),

rs.getString("first\_name"),

rs.getString("last\_name"),

rs.getString("email"),

rs.getString("expertise") // changed from 'department' to 'expertise'

);

}

} catch (SQLException e) {

e.printStackTrace();

}

return null;

}

public List<Teacher> getAllTeachers() {

List<Teacher> teachers = new ArrayList<>();

String query = "SELECT \* FROM teachers";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query);

ResultSet rs = stmt.executeQuery()) {

while (rs.next()) {

Teacher teacher = new Teacher(

rs.getInt("id"),

rs.getString("first\_name"),

rs.getString("last\_name"),

rs.getString("email"),

rs.getString("expertise")

);

teachers.add(teacher);

}

} catch (SQLException e) {

e.printStackTrace();

}

return teachers;

}

public Teacher getTeacherByEmail(String email) {

String query = "SELECT \* FROM teachers WHERE email = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setString(1, email);

ResultSet rs = stmt.executeQuery();

if (rs.next()) {

return new Teacher(

rs.getInt("id"),

rs.getString("first\_name"),

rs.getString("last\_name"),

rs.getString("email"),

rs.getString("expertise")

);

}

} catch (SQLException e) {

e.printStackTrace();

}

return null;

}

public boolean teacherExists(String email) {

String query = "SELECT COUNT(\*) FROM teachers WHERE email = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setString(1, email);

ResultSet rs = stmt.executeQuery();

if (rs.next()) return rs.getInt(1) > 0;

} catch (SQLException e) {

e.printStackTrace();

}

return false;

}

public boolean teacherExists(int id) {

String query = "SELECT COUNT(\*) FROM teachers WHERE id = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, id);

ResultSet rs = stmt.executeQuery();

if (rs.next()) return rs.getInt(1) > 0;

} catch (SQLException e) {

e.printStackTrace();

}

return false;

}

public void insertTeacher(Teacher teacher) {

if (teacherExists(teacher.getEmail())) {

System.out.println("Teacher with this email already exists.");

return;

}

String query = "INSERT INTO teachers (first\_name, last\_name, email, expertise) VALUES (?, ?, ?, ?)";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query, Statement.RETURN\_GENERATED\_KEYS)) {

stmt.setString(1, teacher.getFirstName());

stmt.setString(2, teacher.getLastName());

stmt.setString(3, teacher.getEmail());

stmt.setString(4, teacher.getExpertise());

int affectedRows = stmt.executeUpdate();

if (affectedRows > 0) {

try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {

if (generatedKeys.next()) {

teacher.setTeacherID(generatedKeys.getInt(1));

}

}

System.out.println("Teacher inserted successfully.");

}

} catch (SQLException e) {

e.printStackTrace();

}

}

public void updateTeacher(Teacher teacher) {

String query = "UPDATE teachers SET first\_name = ?, last\_name = ?, email = ?, expertise = ? WHERE id = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setString(1, teacher.getFirstName());

stmt.setString(2, teacher.getLastName());

stmt.setString(3, teacher.getEmail());

stmt.setString(4, teacher.getExpertise());

stmt.setInt(5, teacher.getTeacherID());

int rowsUpdated = stmt.executeUpdate();

if (rowsUpdated > 0) {

System.out.println("Teacher updated successfully.");

} else {

System.out.println("Teacher not found.");

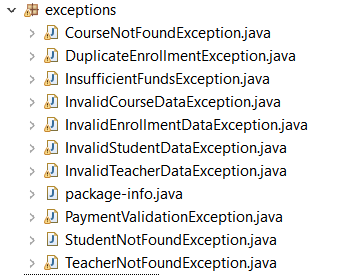
}

} catch (SQLException e) {

e.printStackTrace();

}

}

}  
  
**6)package exceptions**  
  
  
**CourseNotFoundException.java**  
  
package exceptions;

public class CourseNotFoundException extends Exception {

public CourseNotFoundException(String message) {

super(message);

}

}

**DuplicateEnrollmentException.java**  
  
package exceptions;

public class DuplicateEnrollmentException extends Exception {

public DuplicateEnrollmentException(String message) {

super(message);

}

}  
  
**InsufficientFundsException.java**  
  
package exceptions;

public class InsufficientFundsException extends Exception {

public InsufficientFundsException(String message) {

super(message);

}

}

**InvalidCourseDataException.java**

package exceptions;

public class InvalidCourseDataException extends Exception {

public InvalidCourseDataException(String message) {

super(message);

}

}

**InvalidEnrollmentDataException.java**  
  
package exceptions;

public class InvalidEnrollmentDataException extends Exception {

public InvalidEnrollmentDataException(String message) {

super(message);

}

}

**InvalidStudentDataException.java**  
  
package exceptions;

public class InvalidStudentDataException extends Exception {

public InvalidStudentDataException(String message) {

super(message);

}

}

**InvalidTeacherDataException.java**  
  
package exceptions;

public class InvalidTeacherDataException extends Exception {

public InvalidTeacherDataException(String message) {

super(message);

}

}

**PaymentValidationException.java**  
package exceptions;

public class PaymentValidationException extends Exception {

public PaymentValidationException(String message) {

super(message);

}

}

**StudentNotFoundException.java**  
package exceptions;

public class StudentNotFoundException extends Exception {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

public StudentNotFoundException(String message) {

super(message);

}

}

**TeacherNotFoundException.java**  
package exceptions;

public class TeacherNotFoundException extends Exception {

public TeacherNotFoundException(String message) {

super(message);

}

}

**Output Screenshot:**  
