Name: - Dipti Kumari

Assignment :- Student Information System (SIS)

Github repository:- https://github.com/dipti-kumari11/hexawarefilessubmission

Task 1: Define Classes

Define the following classes based on the domain description:

Student class with the following attributes:

• Student ID

• First Name

• Last Name

• Date of Birth

• Email

• Phone Number

CODE:-

package Student;

import java.util.\*;

public class student {

private int studentId;

private String firstName;

private String lastName;

private Date dateOfBirth;

private String email;

private String phoneNumber;

public student(int studentId, String firstName, String lastName, Date dateOfBirth, String email, String phoneNumber) {

this.studentId = studentId;

this.firstName = firstName;

this.lastName = lastName;

this.dateOfBirth = dateOfBirth;

this.email = email;

this.phoneNumber = phoneNumber;

}

public int getStudentId() { return studentId; }

public void setStudentId(int studentId) { this.studentId = studentId; }

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(Date dateOfBirth) { this.dateOfBirth = dateOfBirth; }

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; }

// Optional: You can override toString() for better output formatting

*@Override*

public String toString() {

return "Student ID: " + studentId + "\n" +

"Name: " + firstName + " " + lastName + "\n" +

"Date of Birth: " + dateOfBirth + "\n" +

"Email: " + email + "\n" +

"Phone Number: " + phoneNumber;

}

// Main method to test the class

public static void main(String[] args) {

// Creating a Date object for the student's date of birth

Calendar calendar = Calendar.*getInstance*();

calendar.set(2000, Calendar.***JANUARY***, 1); // Example date: January 1, 2000

Date dob = calendar.getTime();

// Creating a Student object

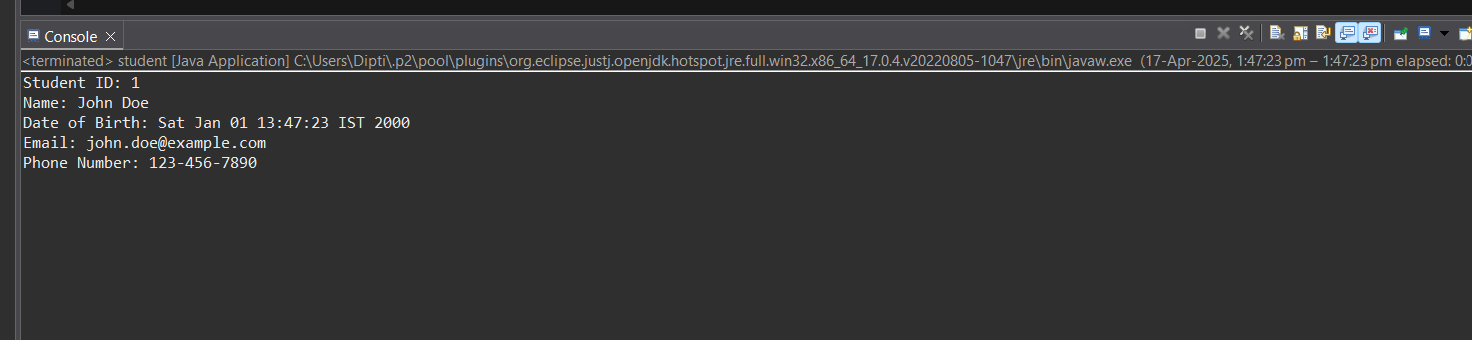
student student = new student(1, "John", "Doe", dob, "john.doe@example.com", "123-456-7890");

// Printing the student details

System.***out***.println(student);

}

}



Course class with the following attributes:

• Course ID

• Course Name

• Course Code

• Instructor Name

CODE:-

package Student;

public class Course {

private int courseId;

private String courseName;

private String courseCode;

private String instructorName;

public Course(int courseId, String courseName, String courseCode, String instructorName) {

this.courseId = courseId;

this.courseName = courseName;

this.courseCode = courseCode;

this.instructorName = instructorName;

}

public int getCourseId() { return courseId; }

public void setCourseId(int courseId) { this.courseId = courseId; }

public String getCourseName() { return courseName; }

public void setCourseName(String courseName) { this.courseName = courseName; }

public String getCourseCode() { return courseCode; }

public void setCourseCode(String courseCode) { this.courseCode = courseCode; }

public String getInstructorName() { return instructorName; }

public void setInstructorName(String instructorName) { this.instructorName = instructorName; }

// Optional: You can override toString() for better output formatting

*@Override*

public String toString() {

return "Course ID: " + courseId + "\n" +

"Course Name: " + courseName + "\n" +

"Course Code: " + courseCode + "\n" +

"Instructor Name: " + instructorName;

}

// Main method to test the class

public static void main(String[] args) {

// Creating a Course object

Course course = new Course(101, "Introduction to Computer Science", "CS101", "Dr. John Doe");

// Printing the course details

System.***out***.println(course);

}

}



Enrollment class to represent the relationship between students and courses. It should have attributes:

• Enrollment ID

• Student ID (reference to a Student)

• Course ID (reference to a Course)

• Enrollment Date

CODE:-

package Student;

import java.util.Date;

public class Enrollment {

private int enrollmentId;

private int studentId;

private int courseId;

private Date enrollmentDate;

public Enrollment(int enrollmentId, int studentId, int courseId, Date enrollmentDate) {

this.enrollmentId = enrollmentId;

this.studentId = studentId;

this.courseId = courseId;

this.enrollmentDate = enrollmentDate;

}

public int getEnrollmentId() { return enrollmentId; }

public void setEnrollmentId(int enrollmentId) { this.enrollmentId = enrollmentId; }

public int getStudentId() { return studentId; }

public void setStudentId(int studentId) { this.studentId = studentId; }

public int getCourseId() { return courseId; }

public void setCourseId(int courseId) { this.courseId = courseId; }

public Date getEnrollmentDate() { return enrollmentDate; }

public void setEnrollmentDate(Date enrollmentDate) { this.enrollmentDate = enrollmentDate; }

// Optional: You can override toString() for better output formatting

@Override

public String toString() {

return "Enrollment ID: " + enrollmentId + "\n" +

"Student ID: " + studentId + "\n" +

"Course ID: " + courseId + "\n" +

"Enrollment Date: " + enrollmentDate;

}

// Main method to test the class

public static void main(String[] args) {

// Creating a Date object for the enrollment date

Date enrollmentDate = new Date(); // Current date and time

// Creating an Enrollment object

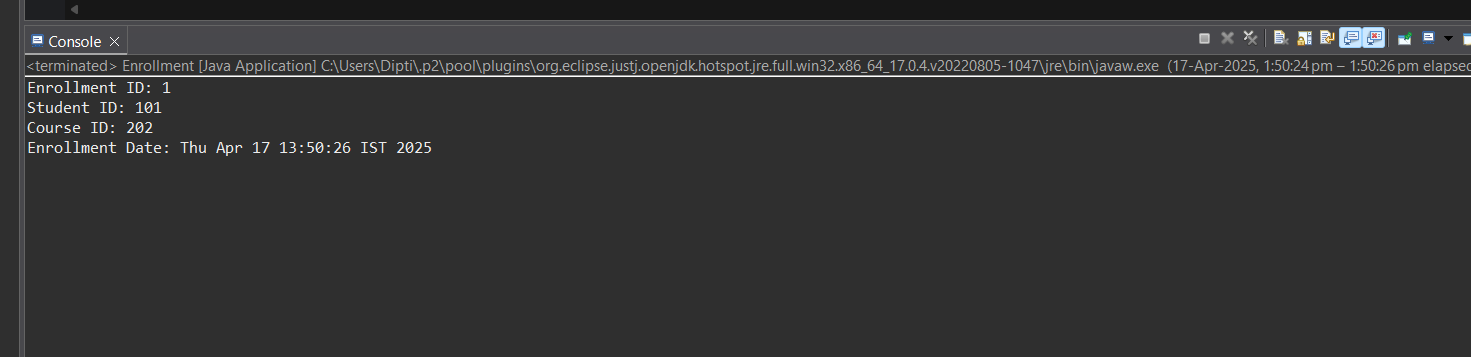
Enrollment enrollment = new Enrollment(1, 101, 202, enrollmentDate);

// Printing the enrollment details

System.out.println(enrollment);

}

}



Teacher class with the following attributes:

• Teacher ID

• First Name

• Last Name

• Email

CODE:-

package Student;

public class Teacher {

private int teacherId;

private String firstName;

private String lastName;

private String email;

public Teacher(int teacherId, String firstName, String lastName, String email) {

this.teacherId = teacherId;

this.firstName = firstName;

this.lastName = lastName;

this.email = email;

}

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; }

// Optional: Override toString() for better output formatting

*@Override*

public String toString() {

return "Teacher ID: " + teacherId + "\n" +

"Name: " + firstName + " " + lastName + "\n" +

"Email: " + email;

}

// Main method to test the class

public static void main(String[] args) {

// Creating a Teacher object

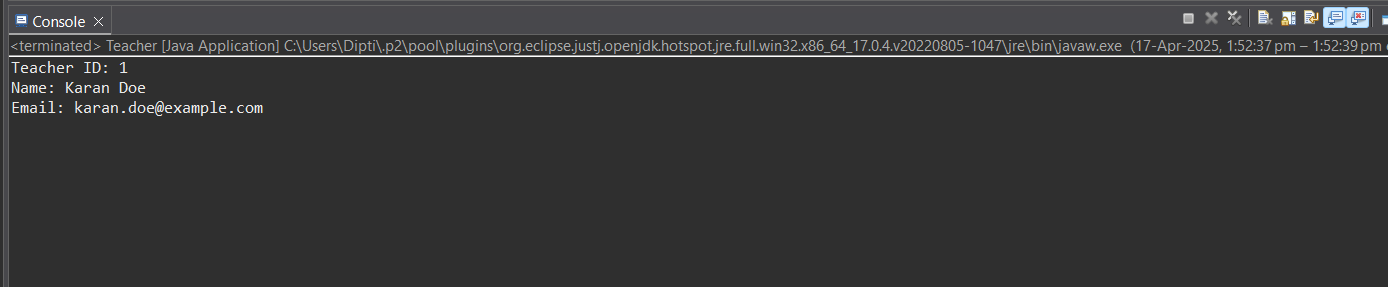
Teacher teacher = new Teacher(1, "Karan", "Doe", "karan.doe@example.com");

// Printing the teacher details

System.***out***.println(teacher);

}

}



Payment class with the following attributes:

• Payment ID

• Student ID (reference to a Student)

• Amount

• Payment Date

CODE:-

package Student;

import java.util.Date;

public class Payment {

private int paymentId;

private int studentId;

private double amount;

private Date paymentDate;

public Payment(int paymentId, int studentId, double amount, Date paymentDate) {

this.paymentId = paymentId;

this.studentId = studentId;

this.amount = amount;

this.paymentDate = paymentDate;

}

public int getPaymentId() { return paymentId; }

public void setPaymentId(int paymentId) { this.paymentId = paymentId; }

public int getStudentId() { return studentId; }

public void setStudentId(int studentId) { this.studentId = studentId; }

public double getAmount() { return amount; }

public void setAmount(double amount) { this.amount = amount; }

public Date getPaymentDate() { return paymentDate; }

public void setPaymentDate(Date paymentDate) { this.paymentDate = paymentDate; }

// Optional: Override toString() for better output formatting

@Override

public String toString() {

return "Payment ID: " + paymentId + "\n" +

"Student ID: " + studentId + "\n" +

"Amount: $" + amount + "\n" +

"Payment Date: " + paymentDate;

}

// Main method to test the class

public static void main(String[] args) {

// Creating a Date object for the payment date

Date paymentDate = new Date(); // Current date and time

// Creating a Payment object

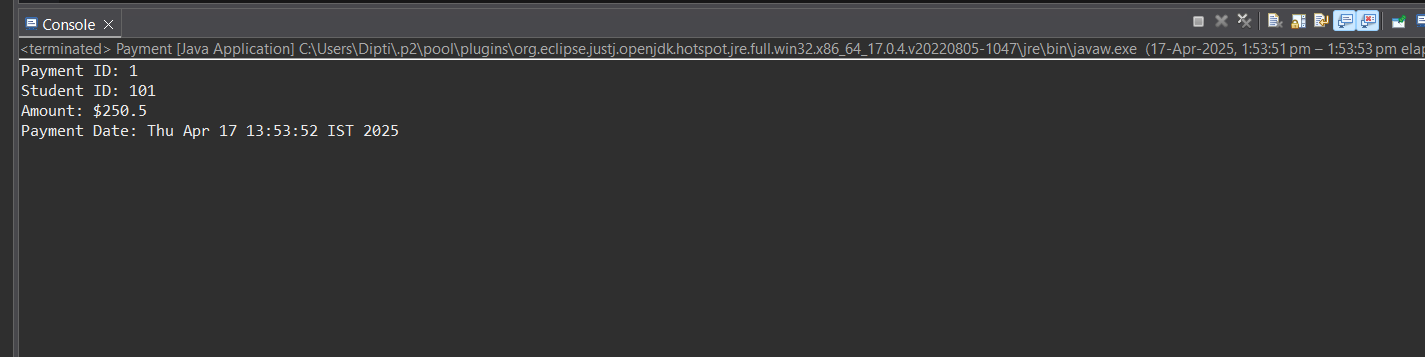
Payment payment = new Payment(1, 101, 250.50, paymentDate);

// Printing the payment details

System.out.println(payment);

}

}



Task 2: Implement Constructors

In same code of task1

Task 3: Implement Methods

In same code of task 1

SIS Class (if you have one to manage interactions):

• EnrollStudentInCourse(student: Student, course: Course): Enrolls a student in a course.

• AssignTeacherToCourse(teacher: Teacher, course: Course): Assigns a teacher to a course.

• RecordPayment(student: Student, amount: decimal, paymentDate: DateTime): Records a

payment made by a student.

• GenerateEnrollmentReport(course: Course): Generates a report of students enrolled in a

specific course.

• GeneratePaymentReport(student: Student): Generates a report of payments made by a specific

student.

• CalculateCourseStatistics(course: Course): Calculates statistics for a specific course, such as the

number of enrollments and total payments.

Use the Methods

In your driver program or any part of your code where you want to perform actions related to the

Student Information System, create instances of your classes, and use the methods you've

implemented.

Repeat this process for using other methods you've implemented in your classes and the SIS class.

CODE:-

// sis/SIS.java

package SIS;

import Student.\*;

import java.util.\*;

public class SIS {

private List<student> students;

private List<Course> courses;

private List<Enrollment> enrollments;

private List<Teacher> teachers;

private List<Payment> payments;

public SIS() {

this.students = new ArrayList<>();

this.courses = new ArrayList<>();

this.enrollments = new ArrayList<>();

this.teachers = new ArrayList<>();

this.payments = new ArrayList<>();

}

public void enrollStudentInCourse(student student, Course course, Date enrollmentDate) {

Enrollment enrollment = new Enrollment(

enrollments.size() + 1,

student.getStudentId(),

course.getCourseId(),

enrollmentDate

);

enrollments.add(enrollment);

}

public void assignTeacherToCourse(Teacher teacher, Course course) {

course.setInstructorName(teacher.getFirstName() + " " + teacher.getLastName());

}

public void recordPayment(student student, double amount, Date paymentDate) {

Payment payment = new Payment(

payments.size() + 1,

student.getStudentId(),

amount,

paymentDate

);

payments.add(payment);

}

public List<student> getStudents() {

return students;

}

public List<Course> getCourses() {

return courses;

}

public List<Enrollment> getEnrollments() {

return enrollments;

}

public List<Teacher> getTeachers() {

return teachers;

}

public List<Payment> getPayments() {

return payments;

}

public void addStudent(student student) {

students.add(student);

}

public void addCourse(Course course) {

courses.add(course);

}

public void addTeacher(Teacher teacher) {

teachers.add(teacher);

}

// Main method to test the SIS functionality

public static void main(String[] args) {

// Creating instances of entities

SIS sis = new SIS();

// Creating students

student student1 = new student(1, "John", "Doe", new Date(2000, 1, 15), "john.doe@example.com", "123-456-7890");

student student2 = new student(2, "Jane", "Smith", new Date(1999, 5, 23), "jane.smith@example.com", "098-765-4321");

// Creating courses

Course course1 = new Course(101, "Mathematics", "MATH101", "");

Course course2 = new Course(102, "Physics", "PHY101", "");

// Creating a teacher

Teacher teacher1 = new Teacher(1, "Dr. Alice", "Johnson", "alice.johnson@example.com");

// Adding students, courses, and teachers to the SIS

sis.addStudent(student1);

sis.addStudent(student2);

sis.addCourse(course1);

sis.addCourse(course2);

sis.addTeacher(teacher1);

// Assigning teacher to a course

sis.assignTeacherToCourse(teacher1, course1);

// Enrolling students in courses

sis.enrollStudentInCourse(student1, course1, new Date());

sis.enrollStudentInCourse(student2, course2, new Date());

// Recording payments

sis.recordPayment(student1, 200.00, new Date());

sis.recordPayment(student2, 150.00, new Date());

// Printing students, courses, enrollments, teachers, and payments

System.out.println("Students:");

for (student s : sis.getStudents()) {

System.out.println(s);

}

System.out.println("\nCourses:");

for (Course c : sis.getCourses()) {

System.out.println(c);

}

System.out.println("\nEnrollments:");

for (Enrollment e : sis.getEnrollments()) {

System.out.println(e);

}

System.out.println("\nTeachers:");

for (Teacher t : sis.getTeachers()) {

System.out.println(t);

}

System.out.println("\nPayments:");

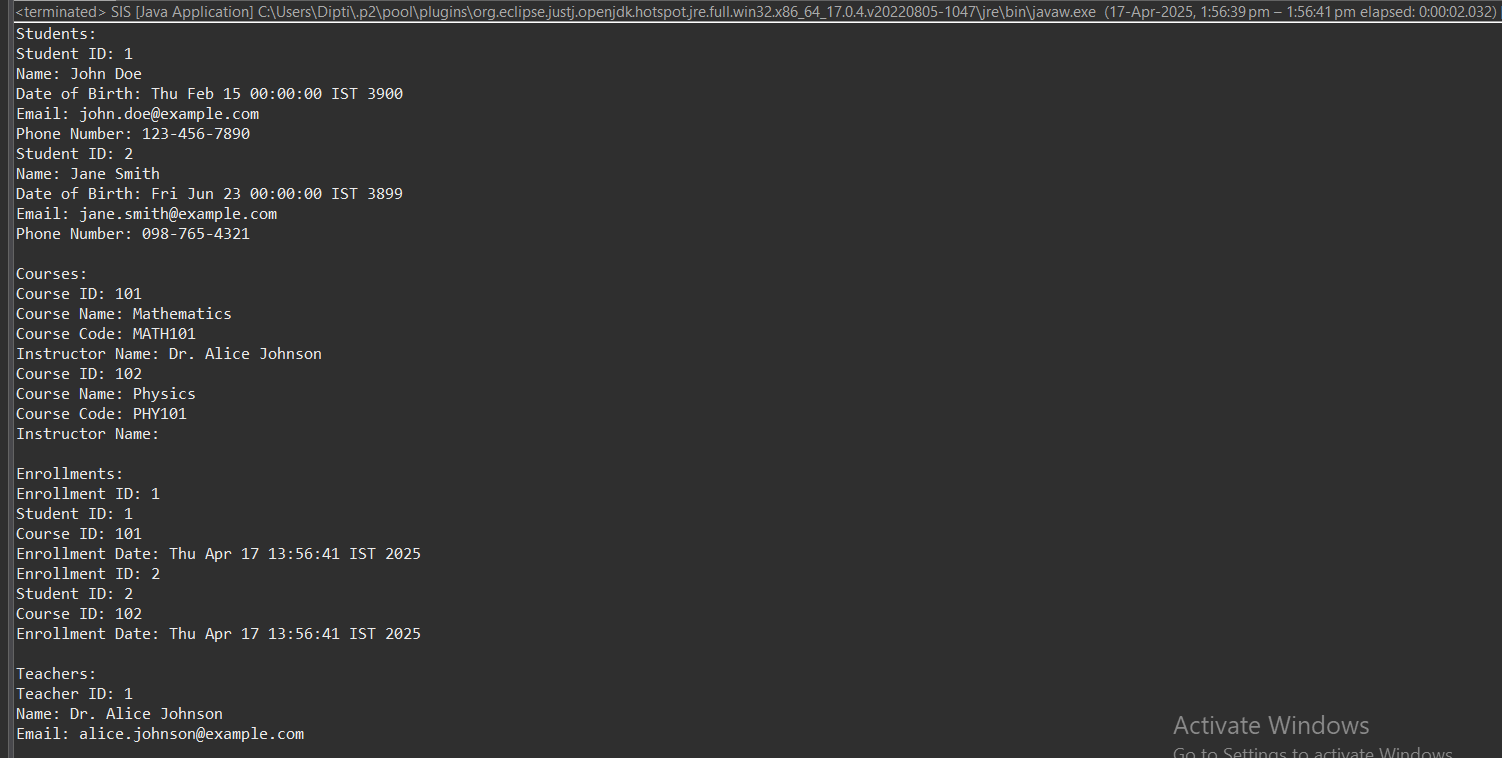
for (Payment p : sis.getPayments()) {

System.out.println(p);

}

}

}





Task 4: Exceptions handling and Custom Exceptions

Implementing custom exceptions allows you to define and throw exceptions tailored to specific

situations or business logic requirements.

Create Custom Exception Classes

You'll need to create custom exception classes that are inherited from the System.Exception class or one

of its derived classes (e.g., System.ApplicationException). These custom exception classes will allow you

to encapsulate specific error scenarios and provide meaningful error messages.

Throw Custom Exceptions

In your code, you can throw custom exceptions when specific conditions or business logic rules are

violated. To throw a custom exception, use the throw keyword followed by an instance of your custom

exception class.

• DuplicateEnrollmentException: Thrown when a student is already enrolled in a course and tries

to enroll again. This exception can be used in the EnrollStudentInCourse method.

• CourseNotFoundException: Thrown when a course does not exist in the system, and you

attempt to perform operations on it (e.g., enrolling a student or assigning a teacher).

• StudentNotFoundException: Thrown when a student does not exist in the system, and you attempt to perform operations on the student (e.g., enrolling in a course, making a payment). • TeacherNotFoundException: Thrown when a teacher does not exist in the system, and you attempt to assign them to a course. • PaymentValidationException: Thrown when there is an issue with payment validation, such as an invalid payment amount or payment date. • InvalidStudentDataException: Thrown when data provided for creating or updating a student is invalid (e.g., invalid date of birth or email format). • InvalidCourseDataException: Thrown when data provided for creating or updating a course is invalid (e.g., invalid course code or instructor name). • InvalidEnrollmentDataException: Thrown when data provided for creating an enrollment is invalid (e.g., missing student or course references). • InvalidTeacherDataException: Thrown when data provided for creating or updating a teacher is invalid (e.g., missing name or email). • InsufficientFundsException: Thrown when a student attempts to enroll in a course but does not have enough funds to make the payment.

CODE:-

// exception/CustomExceptions.java

package StudentException;

public class CustomExceptions {

public static class DuplicateEnrollmentException extends Exception {

public DuplicateEnrollmentException(String message) {

super(message);

}

}

public static class CourseNotFoundException extends Exception {

public CourseNotFoundException(String message) {

super(message);

}

}

public static class StudentNotFoundException extends Exception {

public StudentNotFoundException(String message) {

super(message);

}

}

public static class TeacherNotFoundException extends Exception {

public TeacherNotFoundException(String message) {

super(message);

}

}

public static class PaymentValidationException extends Exception {

public PaymentValidationException(String message) {

super(message);

}

}

public static class InvalidStudentDataException extends Exception {

public InvalidStudentDataException(String message) {

super(message);

}

}

public static class InvalidCourseDataException extends Exception {

public InvalidCourseDataException(String message) {

super(message);

}

}

public static class InvalidEnrollmentDataException extends Exception {

public InvalidEnrollmentDataException(String message) {

super(message);

}

}

public static class InvalidTeacherDataException extends Exception {

public InvalidTeacherDataException(String message) {

super(message);

}

}

public static class InsufficientFundsException extends Exception {

public InsufficientFundsException(String message) {

super(message);

}

}

}

Task 5: Collections

Implement relationships between classes using appropriate data structures (e.g., lists or dictionaries) to

maintain associations between students, courses, enrollments, teachers, and payments.

These relationships are essential for the Student Information System (SIS) to track and manage student

enrollments, teacher assignments, and payments accurately.

Define Class-Level Data Structures

You will need class-level data structures within each class to maintain relationships. Here's how to

define them for each class:

Student Class:

Create a list or collection property to store the student's enrollments. This property will hold references

to Enrollment objects.

Example: List<Enrollment> Enrollments { get; set; }

Course Class:

Create a list or collection property to store the course's enrollments. This property will hold references

to Enrollment objects.

Example: List<Enrollment> Enrollments { get; set; }

Enrollment Class:

Include properties to hold references to both the Student and Course objects.

Example: Student Student { get; set; } and Course Course { get; set; }

Teacher Class:

Create a list or collection property to store the teacher's assigned courses. This property will hold

references to Course objects.

Example: List AssignedCourses { get; set; } Payment Class: Include a property to hold a reference to the Student object. Example: Student Student { get; set; } Update Constructor(s) In the constructors of your classes, initialize the list or collection properties to create empty collections when an object is instantiated. Repeat this for the Course, Teacher, and Payment classes, where applicable.

CODE:- package SIS;

import java.util.\*;

//Student Class

class Student {

String name;

int studentId;

List<Enrollment> enrollments;

public Student(String name, int studentId) {

this.name = name;

this.studentId = studentId;

this.enrollments = new ArrayList<>();

}

public void addEnrollment(Enrollment enrollment) {

enrollments.add(enrollment);

}

*@Override*

public String toString() {

return "Student: " + name + " (ID: " + studentId + ")";

}

}

//Course Class

class Course {

String courseName;

int courseId;

List<Enrollment> enrollments;

public Course(String courseName, int courseId) {

this.courseName = courseName;

this.courseId = courseId;

this.enrollments = new ArrayList<>();

}

public void addEnrollment(Enrollment enrollment) {

enrollments.add(enrollment);

}

*@Override*

public String toString() {

return "Course: " + courseName + " (ID: " + courseId + ")";

}

}

//Enrollment Class

class Enrollment {

Student student;

Course course;

public Enrollment(Student student, Course course) {

this.student = student;

this.course = course;

student.addEnrollment(this);

course.addEnrollment(this);

}

*@Override*

public String toString() {

return "Enrollment -> " + student.name + " in " + course.courseName;

}

}

//Teacher Class

class Teacher {

String name;

int teacherId;

List<Course> assignedCourses;

public Teacher(String name, int teacherId) {

this.name = name;

this.teacherId = teacherId;

this.assignedCourses = new ArrayList<>();

}

public void assignCourse(Course course) {

assignedCourses.add(course);

}

*@Override*

public String toString() {

return "Teacher: " + name + " (ID: " + teacherId + ")";

}

}

//Payment Class

class Payment {

double amount;

Student student;

public Payment(double amount, Student student) {

this.amount = amount;

this.student = student;

}

*@Override*

public String toString() {

return "Payment of $" + amount + " by " + student.name;

}

}

//Main Application

public class SISApp {

public static void main(String[] args) {

// Creating Students

Student s1 = new Student("Alice", 1);

Student s2 = new Student("Bob", 2);

// Creating Courses

Course c1 = new Course("Math", 101);

Course c2 = new Course("Science", 102);

// Creating Teachers

Teacher t1 = new Teacher("Mr. Smith", 501);

t1.assignCourse(c1);

t1.assignCourse(c2);

// Enrolling Students

Enrollment e1 = new Enrollment(s1, c1);

Enrollment e2 = new Enrollment(s2, c2);

Enrollment e3 = new Enrollment(s1, c2);

// Making Payments

Payment p1 = new Payment(500.00, s1);

Payment p2 = new Payment(450.00, s2);

// Output

System.***out***.println(s1);

for (Enrollment e : s1.enrollments) {

System.***out***.println(" " + e);

}

System.***out***.println(t1);

for (Course c : t1.assignedCourses) {

System.***out***.println(" Assigned Course: " + c);

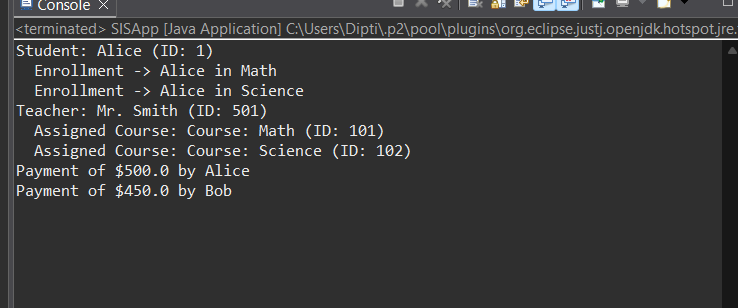
}

System.***out***.println(p1);

System.***out***.println(p2);

}

}



Task 6: Create Methods for Managing Relationships

To add, remove, or retrieve related objects, you should create methods within your SIS class or each

relevant class.

• AddEnrollment(student, course, enrollmentDate): In the SIS class, create a method that adds an

enrollment to both the Student's and Course's enrollment lists. Ensure the Enrollment object

references the correct Student and Course.

• AssignCourseToTeacher(course, teacher): In the SIS class, create a method to assign a course to

a teacher. Add the course to the teacher's AssignedCourses list.

• AddPayment(student, amount, paymentDate): In the SIS class, create a method that adds a

payment to the Student's payment history. Ensure the Payment object references the correct

Student.

• GetEnrollmentsForStudent(student): In the SIS class, create a method to retrieve all enrollments

for a specific student.

• GetCoursesForTeacher(teacher): In the SIS class, create a method to retrieve all courses

assigned to a specific teacher.

Create a Driver Program

A driver program (also known as a test program or main program) is essential for testing and

demonstrating the functionality of your classes and methods within your Student Information System

(SIS) assignment. In this task, you will create a console application that serves as the entry point for your

SIS and allows you to interact with and test your implemented classes and methods.

Add References to Your SIS Classes

Ensure that your SIS classes (Student, Course, Enrollment, Teacher, Payment) and the SIS class (if you

have one to manage interactions) are defined in separate files within your project or are referenced

properly.

If you have defined these classes in separate files, make sure to include using statements in your driver

program to access them:

Implement the Main Method

In the console application, the Main method serves as the entry point for your program. This is where

you will create instances of your classes, call methods, and interact with your Student Information

System.

In the Main method, you create instances of your classes (e.g., Student, Course, and SIS) and then

interact with your Student Information System by calling methods and handling exceptions.

CODE:-

// main/MainModuleStudent.java

package mainStudent;

import Student.student;

import Student.Course;

import Student.Teacher;

import Student.Enrollment;

import SIS.SIS;

import StudentException.CustomExceptions.\*;

import java.text.SimpleDateFormat;

import java.util.\*;

public class MainModuleStudent {

private static final Scanner ***scanner*** = new Scanner(System.***in***);

private static final SIS ***sis*** = new SIS();

public static void main(String[] args) {

int choice;

do {

System.***out***.println("\n===== Student Information System =====");

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

System.***out***.println("2. Enroll Student in Course");

System.***out***.println("3. Assign Teacher to Course");

System.***out***.println("4. Record Payment");

System.***out***.println("5. View Enrollments");

System.***out***.println("6. Exit");

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

try {

choice = Integer.*parseInt*(***scanner***.nextLine());

switch (choice) {

case 1 -> *addStudent*();

case 2 -> *enrollStudent*();

case 3 -> *assignTeacher*();

case 4 -> *recordPayment*();

case 5 -> *viewEnrollments*();

case 6 -> System.***out***.println("Exiting SIS. Goodbye!");

default -> System.***out***.println("Invalid choice. Try again.");

}

} catch (DuplicateEnrollmentException | StudentNotFoundException | CourseNotFoundException |

TeacherNotFoundException | PaymentValidationException | InvalidStudentDataException |

InvalidTeacherDataException e) {

System.***out***.println("Custom Exception: " + e.getMessage());

choice = 0;

} catch (Exception e) {

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

choice = 0;

}

} while (choice != 6);

}

private static void addStudent() throws InvalidStudentDataException {

try {

System.***out***.print("Enter Student ID: ");

int id = Integer.*parseInt*(***scanner***.nextLine());

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

String firstName = ***scanner***.nextLine();

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

String lastName = ***scanner***.nextLine();

System.***out***.print("Date of Birth (yyyy-MM-dd): ");

Date dob = new SimpleDateFormat("yyyy-MM-dd").parse(***scanner***.nextLine());

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

String email = ***scanner***.nextLine();

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

String phone = ***scanner***.nextLine();

if (email.isEmpty() || !email.contains("@")) {

throw new InvalidStudentDataException("Invalid email format.");

}

student student = new student(id, firstName, lastName, dob, email, phone);

***sis***.addStudent(student);

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

} catch (InvalidStudentDataException e) {

throw e;

} catch (Exception e) {

throw new InvalidStudentDataException("Error adding student: " + e.getMessage());

}

}

private static void enrollStudent() throws StudentNotFoundException, CourseNotFoundException, DuplicateEnrollmentException {

System.***out***.print("Enter Student ID: ");

int sid = Integer.*parseInt*(***scanner***.nextLine());

student student = ***sis***.getStudents().stream()

.filter(s -> s.getStudentId() == sid)

.findFirst()

.orElseThrow(() -> new StudentNotFoundException("Student not found."));

System.***out***.print("Enter Course ID: ");

int cid = Integer.*parseInt*(***scanner***.nextLine());

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

String cname = ***scanner***.nextLine();

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

String ccode = ***scanner***.nextLine();

System.***out***.print("Instructor Name: ");

String instructor = ***scanner***.nextLine();

Course course = new Course(cid, cname, ccode, instructor);

***sis***.addCourse(course);

boolean alreadyEnrolled = ***sis***.getEnrollments().stream()

.anyMatch(e -> e.getStudentId() == sid && e.getCourseId() == cid);

if (alreadyEnrolled) {

throw new DuplicateEnrollmentException("Student is already enrolled in this course.");

}

***sis***.enrollStudentInCourse(student, course, new Date());

System.***out***.println("Student enrolled successfully.");

}

private static void assignTeacher() throws CourseNotFoundException, InvalidTeacherDataException, TeacherNotFoundException {

System.***out***.print("Enter Teacher ID: ");

int tid = Integer.*parseInt*(***scanner***.nextLine());

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

String fname = ***scanner***.nextLine();

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

String lname = ***scanner***.nextLine();

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

String email = ***scanner***.nextLine();

if (email.isEmpty() || !email.contains("@")) {

throw new InvalidTeacherDataException("Invalid email for teacher.");

}

// Optional: Check if teacher already exists

Optional<Teacher> existingTeacher = ***sis***.getTeachers().stream()

.filter(t -> t.getTeacherId() == tid)

.findFirst();

Teacher teacher;

if (existingTeacher.isPresent()) {

teacher = existingTeacher.get();

} else {

teacher = new Teacher(tid, fname, lname, email);

***sis***.addTeacher(teacher);

}

System.***out***.print("Enter Course ID to assign: ");

int cid = Integer.*parseInt*(***scanner***.nextLine());

Course course = ***sis***.getCourses().stream()

.filter(c -> c.getCourseId() == cid)

.findFirst()

.orElseThrow(() -> new CourseNotFoundException("Course not found."));

***sis***.assignTeacherToCourse(teacher, course);

System.***out***.println("Teacher assigned to course.");

}

private static void recordPayment() throws StudentNotFoundException, PaymentValidationException {

System.***out***.print("Enter Student ID: ");

int sid = Integer.*parseInt*(***scanner***.nextLine());

System.***out***.print("Enter Payment Amount: ");

double amount = Double.*parseDouble*(***scanner***.nextLine());

System.***out***.print("Enter Payment Date (yyyy-MM-dd): ");

Date date;

try {

date = new SimpleDateFormat("yyyy-MM-dd").parse(***scanner***.nextLine());

} catch (Exception e) {

throw new PaymentValidationException("Invalid payment date.");

}

if (amount <= 0) {

throw new PaymentValidationException("Payment amount must be positive.");

}

student student = ***sis***.getStudents().stream()

.filter(s -> s.getStudentId() == sid)

.findFirst()

.orElseThrow(() -> new StudentNotFoundException("Student not found."));

***sis***.recordPayment(student, amount, date);

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

}

private static void viewEnrollments() {

for (Enrollment e : ***sis***.getEnrollments()) {

System.***out***.println("Enrollment ID: " + e.getEnrollmentId() +

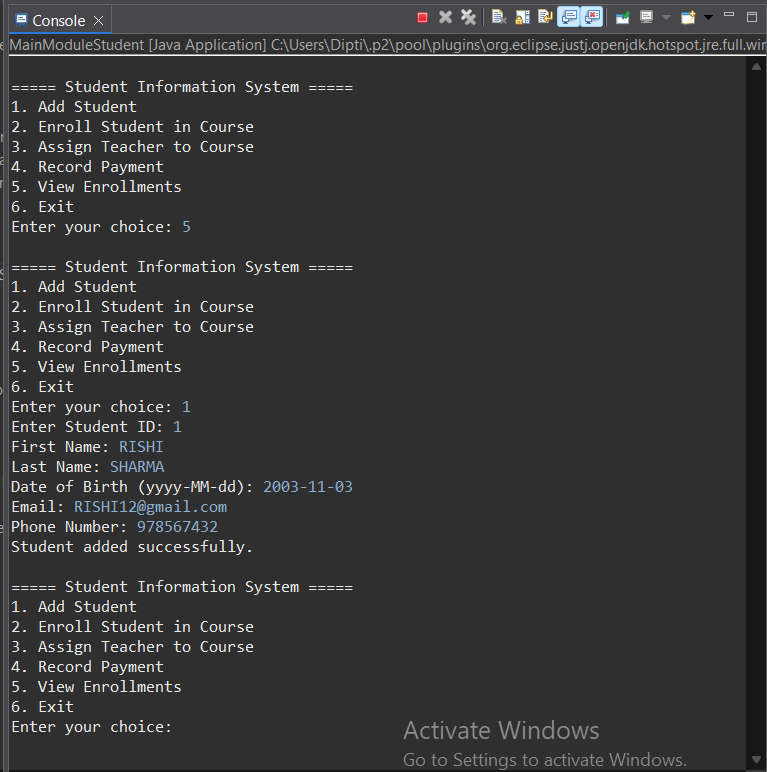
", Student ID: " + e.getStudentId() +

", Course ID: " + e.getCourseId());

}

}

}



Task 7: Database Connectivity

Database Initialization:

Implement a method that initializes a database connection and creates tables for storing student,

course, enrollment, teacher, and payment information. Create SQL scripts or use code-first migration to

create tables with appropriate schemas for your SIS.

Data Retrieval:

Implement methods to retrieve data from the database. Users should be able to request information

about students, courses, enrollments, teachers, or payments. Ensure that the data retrieval methods

handle exceptions and edge cases gracefully.

Data Insertion and Updating:

Implement methods to insert new data (e.g., enrollments, payments) into the database and update

existing data (e.g., student information). Use methods to perform data insertion and updating.

Implement validation checks to ensure data integrity and handle any errors during these operations.

Transaction Management:

Implement methods for handling database transactions when enrolling students, assigning teachers, or

recording payments. Transactions should be atomic and maintain data integrity. Use database

transactions to ensure that multiple related operations either all succeed or all fail. Implement error

handling and rollback mechanisms in case of transaction failures.

Dynamic Query Builder:

Implement a dynamic query builder that allows users to construct and execute custom SQL queries to

retrieve specific data from the database. Users should be able to specify columns, conditions, and

sorting criteria. Create a query builder method that dynamically generates SQL queries based on user

input. Implement parameterization and sanitation of user inputs to prevent SQL injection.

CODE:-

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.Statement;

import java.sql.ResultSet;

public class insertRecords {

public static void main(String[] args) {

try {

Class.*forName*("com.mysql.cj.jdbc.Driver");

final String url = "jdbc:mysql:///SISDB";

final String user = "root";

final String password = "root";

Connection con = DriverManager.*getConnection*(url, user, password);

Statement st = con.createStatement();

String createTable = """

CREATE TABLE IF NOT EXISTS students (

student\_id INT PRIMARY KEY,

first\_name VARCHAR(20),

last\_name VARCHAR(30),

dob DATE,

email VARCHAR(50),

phone\_no VARCHAR(15)

);

""";

st.execute(createTable);

System.***out***.println("Students table checked/created.");

String insertQuery = """

INSERT INTO students

VALUES (12, 'Deepa', 'Kumari', '2004-10-23', 'deepachandra900@gmail.com', '874794672');

""";

int inserted = st.executeUpdate(insertQuery);

if (inserted == 1) {

System.***out***.println("Record inserted.");

} else {

System.***out***.println(" Record not inserted.");

}

String selectQuery = "SELECT \* FROM students;";

ResultSet rs = st.executeQuery(selectQuery);

System.***out***.println("\Student Records:");

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

while (rs.next()) {

System.***out***.println("ID: " + rs.getInt("student\_id") +

" | Name: " + rs.getString("first\_name") + " " + rs.getString("last\_name") +

" | DOB: " + rs.getDate("dob") +

" | Email: " + rs.getString("email") +

" | Phone: " + rs.getString("phone\_no"));

}

rs.close();

st.close();

con.close();

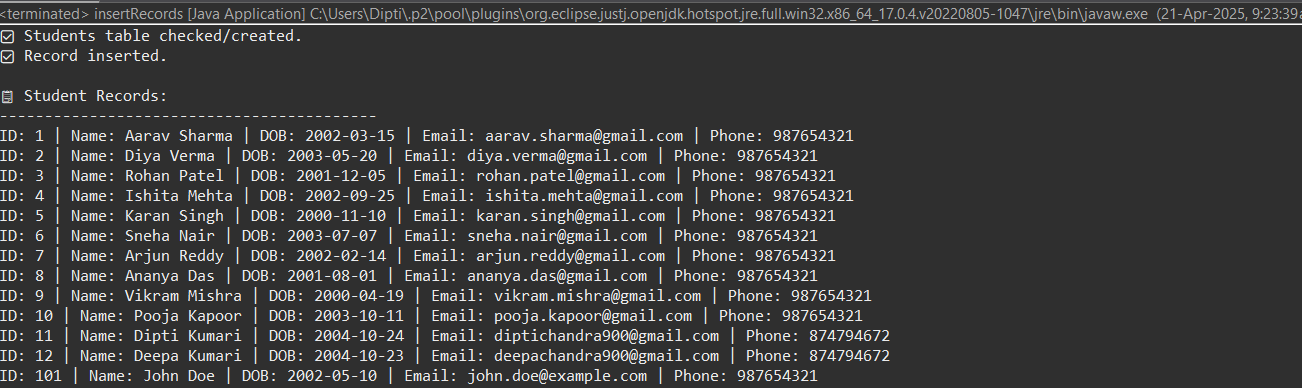
} catch (Exception e) {

e.printStackTrace();

}

}

}



Task 8: Student Enrollment

In this task, a new student, John Doe, is enrolling in the SIS. The system needs to record John's

information, including his personal details, and enroll him in a few courses. Database connectivity is

required to store this information.

John Doe's details:

• First Name: John

• Last Name: Doe

• Date of Birth: 1995-08-15

• Email: john.doe@example.com

• Phone Number: 123-456-7890

John is enrolling in the following courses:

• Course 1: Introduction to Programming

• Course 2: Mathematics 101

The system should perform the following tasks:

• Create a new student record in the database.

• Enroll John in the specified courses by creating enrollment records in the database.

CODE:-

package com.sis.model;

public class Main {

public static void main(String[] args) {

DBUtil.initializeDatabase(); // Optional: Run only once to create tables

SISDao dao = new SISDao();

// Step 1: Insert John Doe

dao.insertStudent(1001, "John", "Doe", "1995-08-15", "john.doe@example.com",

"123-456-7890");

// Step 2: Insert courses

dao.insertCourse(2001, "Introduction to Programming", "CS101", "Dr. Smith",

3000.00);

dao.insertCourse(2002, "Mathematics 101", "MATH101", "Prof. Allen",

2800.00);

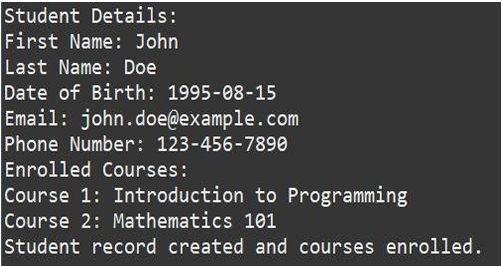
// Step 3: Enroll John in both courses

dao.enrollStudent(3001, 1001, 2001, "2025-04-15");

dao.enrollStudent(3002, 1001, 2002, "2025-04-15");

}

}



Task 9: Teacher Assignment

In this task, a new teacher, Sarah Smith, is assigned to teach a course. The system needs to update the

course record to reflect the teacher assignment.

Teacher's Details:

• Name: Sarah Smith

• Email: sarah.smith@example.com

• Expertise: Computer Science

Course to be assigned:

• Course Name: Advanced Database Management

• Course Code: CS302

The system should perform the following tasks:

• Retrieve the course record from the database based on the course code.

• Assign Sarah Smith as the instructor for the course.

• Update the course record in the database with the new instructor information.

public class Main {

public static void main(String[] args) {

SISDao dao = new SISDao();

// Step 1: Insert teacher

dao.insertTeacher("Sarah", "sarah.smith@example.com", "Computer Science");

// Step 2: Insert course

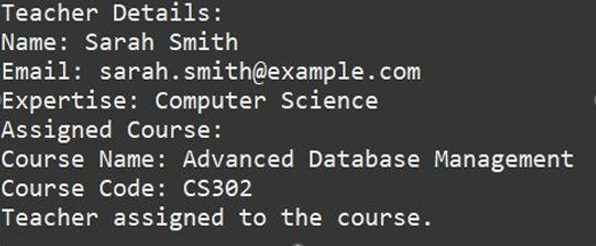
dao.insertCourse(10, "Advanced Database Management", "CS302", "", 5000.0);

// Step 3: Assign teacher to course

dao.assignTeacherToCourse("CS302", "Sarah Smith");

}

}



Task 10: Payment Record

In this task, a student, Jane Johnson, makes a payment for her enrolled courses. The system needs to

record this payment in the database.

Jane Johnson's details:

• Student ID: 101

• Payment Amount: $500.00

• Payment Date: 2023-04-10

The system should perform the following tasks:

• Retrieve Jane Johnson's student record from the database based on her student ID.

• Record the payment information in the database, associating it with Jane's student record.

• Update Jane's outstanding balance in the database based on the payment amount.

public void recordPayment(int studentId, double amount, String paymentDate)

{

String insertPayment = "INSERT INTO Payments (studentId, amount,

paymentDate) VALUES (?, ?, ?)";

String updateBalance = "UPDATE Students SET balance = balance - ? WHERE id

= ?";

try (Connection conn = DBUtil.getConnection()) {

conn.setAutoCommit(false);

try (PreparedStatement ps1 = conn.prepareStatement(insertPayment);

PreparedStatement ps2 = conn.prepareStatement(updateBalance)) {

ps1.setInt(1, studentId);

ps1.setDouble(2, amount);

ps1.setDate(3, Date.valueOf(paymentDate));

ps1.executeUpdate();

ps2.setDouble(1, amount);

ps2.setInt(2, studentId);

ps2.executeUpdate();

conn.commit();

System.out.println("Payment recorded and balance updated.");

} catch (SQLException e) {

conn.rollback();

System.out.println("Payment failed: " + e.getMessage());

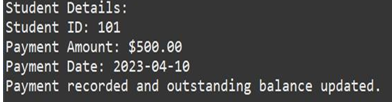
}

} catch (SQLException e) {

e.printStackTrace();

}

}



Task 11: Enrollment Report Generation

In this task, an administrator requests an enrollment report for a specific

course, "Computer Science 101." The system needs to retrieve enrollment

information from the database and generate a report. Course to generate

the report for:

• Course Name: Computer Science 101

The system should perform the following tasks:

• Retrieve enrollment records from the database for the specified course.

• Generate an enrollment report listing all students enrolled in Computer

Science 101.

• Display or save the report for the administrator.

public class Main {

public static void main(String[] args) {

DBUtil.initializeDatabase();

SISDao dao = new SISDao();

dao.generateEnrollmentReport("Computer Science 101");

}

}

