Atharva Padawale – SIS assignment

Student Information System (SIS) - Java

Name: **Atharva Padawale**

Github: https://github.com/AtharvaPadawale/hexaware-Java-Batch-5

Implement OOPs

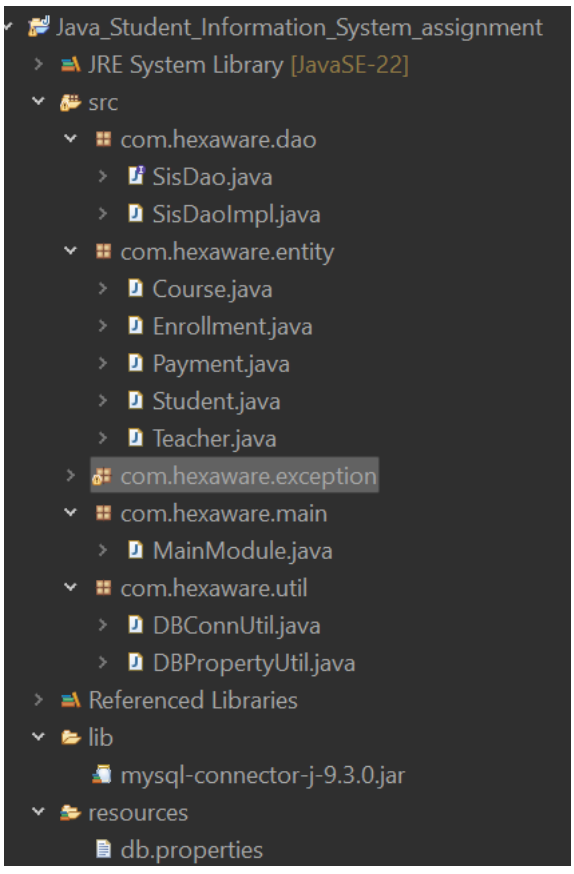
A Student Information System (SIS) manages information about students, courses, student enrollments, teachers, and payments. Each student can enroll in multiple courses, each course can have multiple students, each course is taught by a teacher, and students make payments for their courses. Students have attributes such as name, date of birth, email, and phone number. Courses have attributes such as course name, course code, and instructor name. Enrollments track which students are enrolled in which courses. Teachers have attributes such as names and email. Payments track the amount and date of payments made by students.

Task 1: Define Classes :

Student class with the following attributes:

- Student ID
- First Name
- Last Name
- Date of Birth
- Email
- Phone Number

```java
public class Student {
    private int studentId;
    private String firstName;
    private String lastName;
    private LocalDate dateOfBirth;
    private String email;
    private String phoneNumber;
    private List<Course> enrolledCourses;
    private List<Payment> paymentHistory;
```



Course class with the following attributes:

- Course ID
- Course Name
- Course Code
- Instructor Name

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

Teacher class with the following attributes:

- Teacher ID
- First Name
- Last Name
- Email

Atharva Padawale – SIS assignment

Payment class with the following attributes:

- Payment ID
- Student ID (reference to a Student)
- Amount
- Payment Date

```
public class Payment {
    private int paymentId;
    private int studentId;
    private double amount;
    private LocalDate paymentDate;
```

Task 2: Implement Constructors

Implement constructors for each class to initialize their attributes. Constructors are special methods that are called when an object of a class is created. They are used to set initial values for the attributes of the class. Below are detailed instructions on how to implement constructors for each class in your (SIS) assignment:

Student Class Constructor

In the Student class, you need to create a constructor that initializes the attributes of a student when an instance of the Student class is created.

```
// Constructor
public Student(int studentId, String firstName, String lastName,
        LocalDate dateOfBirth, String email, String phoneNumber) {
    this.studentId = studentId;
    this.firstName = firstName;
    this.lastName = lastName;
    this.dateOfBirth = dateOfBirth;
    this.email = email;
    this.phoneNumber = phoneNumber;
    this.enrolledCourses = new ArrayList<>();
    this.paymentHistory = new ArrayList<>();
}
```

SIS Class Constructor

If you have a class that represents the Student Information System itself (e.g., SIS class), you may also implement a constructor for it. This constructor can be used to set up any initial configuration for the SIS. Repeat the above process for each class Course, Enrollment, Teacher, Payment by defining constructors that initialize their respective attributes.

Task 3: Implement Methods

Implement methods in your classes to perform various operations related to the Student Information System (SIS). These methods will allow you to interact with and manipulate data within your system. Below are detailed instructions on how to implement methods in each class: Implement the following methods in the appropriate classes:

Student Class:

- EnrollInCourse(course: Course): Enrolls the student in a course.
- UpdateStudentInfo(firstName: string, lastName: string, dateOfBirth: DateTime, email: string,
- phoneNumber: string): Updates the student's information.
- MakePayment(amount: decimal, paymentDate: DateTime): Records a payment made by the
- student.
- DisplayStudentInfo(): Displays detailed information about the student.

- GetEnrolledCourses(): Retrieves a list of courses in which the student is enrolled.
- GetPaymentHistory(): Retrieves a list of payment records for the student.

```
// Enroll student in a course
public void enrollInCourse(Course course) {
    enrolledCourses.add(course);
}

// Update student information
public void updateStudentInfo(String firstName, String lastName, LocalDate dateOfBirth, String email, String phoneNumber) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.dateOfBirth = dateOfBirth;
    this.email = email;
    this.phoneNumber = phoneNumber;
}

// Record a payment
public void makePayment(double amount, LocalDate paymentDate) {
    Payment payment = new Payment(paymentHistory.size() + 1, this.studentId, amount, paymentDate);
    paymentHistory.add(payment);
}
```

Course Class:

- AssignTeacher(teacher: Teacher): Assigns a teacher to the course.
- UpdateCourseInfo(courseCode: string, courseName: string, instructor: string): Updates course
- information.
- DisplayCourseInfo(): Displays detailed information about the course.
- GetEnrollments(): Retrieves a list of student enrollments for the course.
- GetTeacher(): Retrieves the assigned teacher for the course.

```
// Assign a teacher
public void assignTeacher(Teacher teacher) {
    this.teacher = teacher;
}

// Update course info
public void updateCourseInfo(String courseCode, String courseName, String instructorName) {
    this.courseCode = courseCode;
    this.courseName = courseName;
    this.instructorName = instructorName;
}

// Display course info
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 Name: " + instructorName);
}

// Get list of enrollments
public List<Enrollment> getEnrollments() {
    return enrollments;
}
```

Teacher Class:

- UpdateTeacherInfo(name: string, email: string, expertise: string): Updates teacher information.
- DisplayTeacherInfo(): Displays detailed information about the teacher.
- GetAssignedCourses(): Retrieves a list of courses assigned to the teacher.

```java
// Update teacher info
public void updateTeacherInfo(String firstName, String lastName, String email) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.email = email;
}

// Display teacher info
public void displayTeacherInfo() {
    System.out.println("Teacher ID: " + teacherId);
    System.out.println("Name: " + firstName + " " + lastName);
    System.out.println("Email: " + email);
}

// Get assigned courses
public List<Course> getAssignedCourses() {
    return assignedCourses;
}
```

Enrollment Class:

- GetStudent(): Retrieves the student associated with the enrollment.
- GetCourse(): Retrieves the course associated with the enrollment.

Payment Class:

- GetStudent(): Retrieves the student associated with the payment.
- GetPaymentAmount(): Retrieves the payment amount.
- GetPaymentDate(): Retrieves the payment date.

```java
public double getAmount() {
    return amount;
```

```java
public LocalDate getPaymentDate() {
    return paymentDate;
}
```

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

- 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.

Task 5: Collections

Implement 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; }

```
// Get list of enrolled courses
public List<Course> getEnrolledCourses() {
    return enrolledCourses;
}
```

Course Class:

Atharva Padawale – SIS assignment

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; }

```
// Get list of enrollments
public List<Enrollment> getEnrollments() {
    return enrollments;
}
```

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<Course> AssignedCourses { get; set; }

```
// Get assigned courses
public List<Course> getAssignedCourses() {
    return assignedCourses;
}
```

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.

```
private List<Course> enrolledCourses;
private List<Payment> paymentHistory;
```

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.

```
public void addEnrollment(Student student, Course course, Date enrollmentDate) throws DuplicateEnrollmentException, InvalidE
    for (Enrollment enrollment : enrollments) {
        if (enrollment.getStudent().getStudentId() == student.getStudentId() && enrollment.getCourse().getCourseId() == cour
            throw new DuplicateEnrollmentException("Student " + student.getFirstName() + " is already enrolled in this cours
        }
    }
    Enrollment newEnrollment = new Enrollment(student, course, new java.sql.Date(enrollmentDate.getTime()));
    enrollments.add(newEnrollment);
    student.enrollInCourse(course);
    course.getEnrollments().add(newEnrollment);
    System.out.println("Enrollment added for student " + student.getFirstName() + " in course " + course.getCourseName());
}
```

- 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.

```
public void assignCourseToTeacher(Course course, Teacher teacher) throws TeacherNotFoundException {
    if (teacher == null) {
        throw new TeacherNotFoundException("Teacher not found.");
    }
    course.assignTeacher(teacher);
    teacher.getAssignedCourses().add(course);
    System.out.println("Course " + course.getCourseName() + " has been assigned to teacher " + teacher.getFirstName());
}
```

- 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.

```
public void addPayment(Student student, double amount, Date paymentDate) throws PaymentValidationException {
    if (amount <= 0) {
        throw new PaymentValidationException("Payment amount must be greater than zero.");
    }
    Payment payment = new Payment(payments.size() + 1, student.getStudentId(), amount, paymentDate);
    payments.add(payment);
    student.makePayment(amount, paymentDate);
    System.out.println("Payment of " + amount + " has been recorded for student " + student.getFirstName());
}
```

- GetEnrollmentsForStudent(student): In the SIS class, create a method to retrieve all enrollments for a specific student.

```
public List<Enrollment> getEnrollmentsForStudent(Student student) {
    List<Enrollment> studentEnrollments = new ArrayList<>();
    for (Enrollment enrollment : enrollments) {
        if (enrollment.getStudent().getStudentId() == student.getStudentId()) {
            studentEnrollments.add(enrollment);
        }
    }
    return studentEnrollments;
}
```

- GetCoursesForTeacher(teacher): In the SIS class, create a method to retrieve all courses assigned to a specific teacher.

```
public void assignCourseToTeacher(Course course, Teacher teacher) throws TeacherNotFoundException {
    if (teacher == null) {
        throw new TeacherNotFoundException("Teacher not found.");
    }
    course.assignTeacher(teacher);
    teacher.getAssignedCourses().add(course);
    System.out.println("Course " + course.getCourseName() + " has been assigned to teacher " + teacher.getFirstName());
}
```

Create a Driver Program

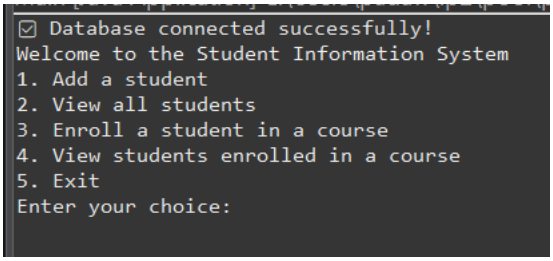
Implement the Main Method

Task 7: Database Connectivity

Database Initialization:

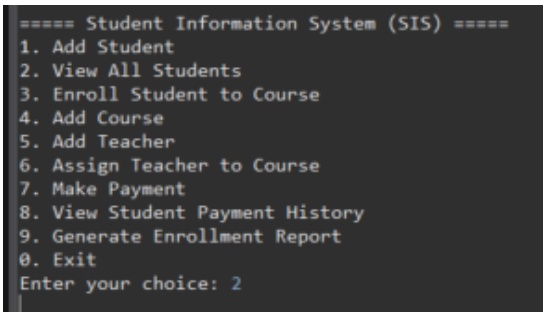
Implement a method that initializes a database connection and creates tables for storing student, course, enrollment, teacher, and payment information. Create SQL scripts or use code-first migration to create tables with appropriate schemas for your SIS.

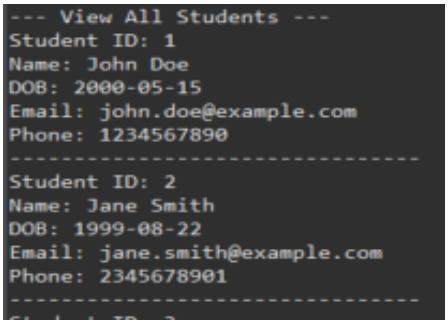
Atharva Padawale – SIS assignment



Data Retrieval:

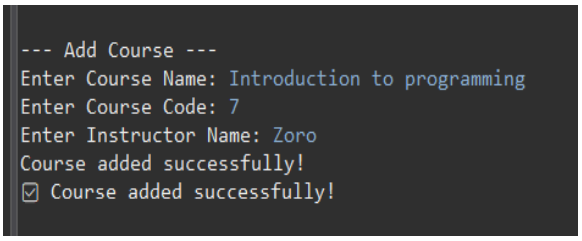
Implement methods to retrieve data from the database. Users should be able to request information about students, courses, enrollments, teachers, or payments. Ensure that the data retrieval methods handle exceptions and edge cases gracefully.

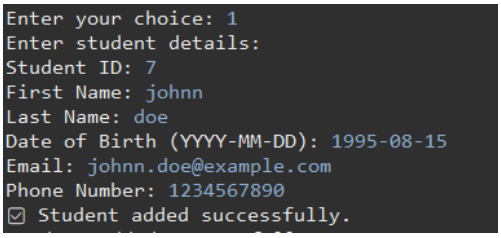




Data Insertion and Updating:

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.

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

Atharva Padawale – SIS assignment

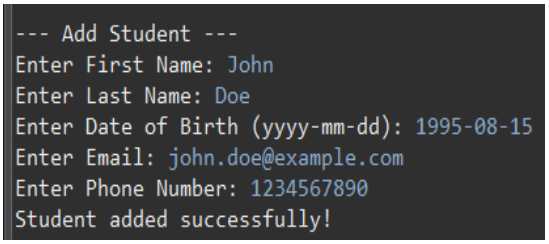
- Date of Birth: 1995-08-15
- Email: john.doe@example.com
- Phone Number: 123-456-7890

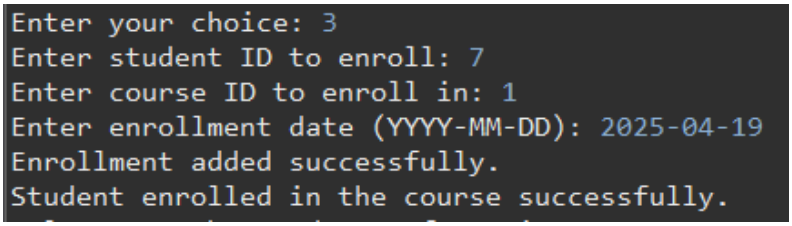
John is enrolling in the following courses:

- 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.





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:

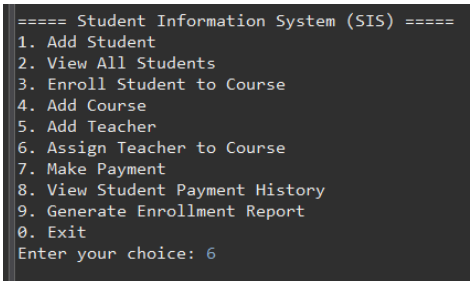
- Name: Sarah Smith
- Email: sarah.smith@example.com
- Expertise: Computer Science

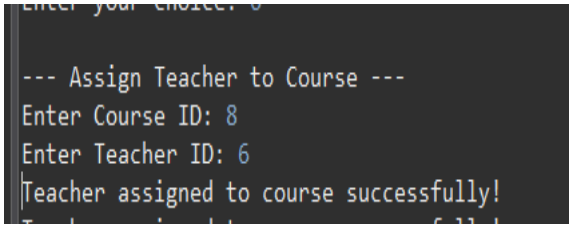
Course to be assigned:

- 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.





Task 10: Payment Record

Atharva Padawale – SIS assignment

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.

```
===== Student Information System (SIS) =====
1. Add Student
2. View All Students
3. Enroll Student to Course
4. Add Course
5. Add Teacher
6. Assign Teacher to Course
7. Make Payment
8. View Student Payment History
9. Generate Enrollment Report
0. Exit
Enter your choice: 7
```

```
--- Make Payment ---
Enter Student ID: 9
Enter Amount: 500
Enter Payment Date (yyyy-mm-dd): -2023-04-10
Payment recorded successfully!
```

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.

```
===== Student Information System (SIS) =====
1. Add Student
2. View All Students
3. Enroll Student to Course
4. Add Course
5. Add Teacher
6. Assign Teacher to Course
7. Make Payment
8. View Student Payment History
9. Generate Enrollment Report
0. Exit
Enter your choice: 9
```

```
--- Generate Enrollment Report ---
Enter Course Id: 3

===== Enrollment Report =====
Course Name: Computer Science 101

Enrolled Students:
Student ID: 3
Name        : Alice Johnson
```