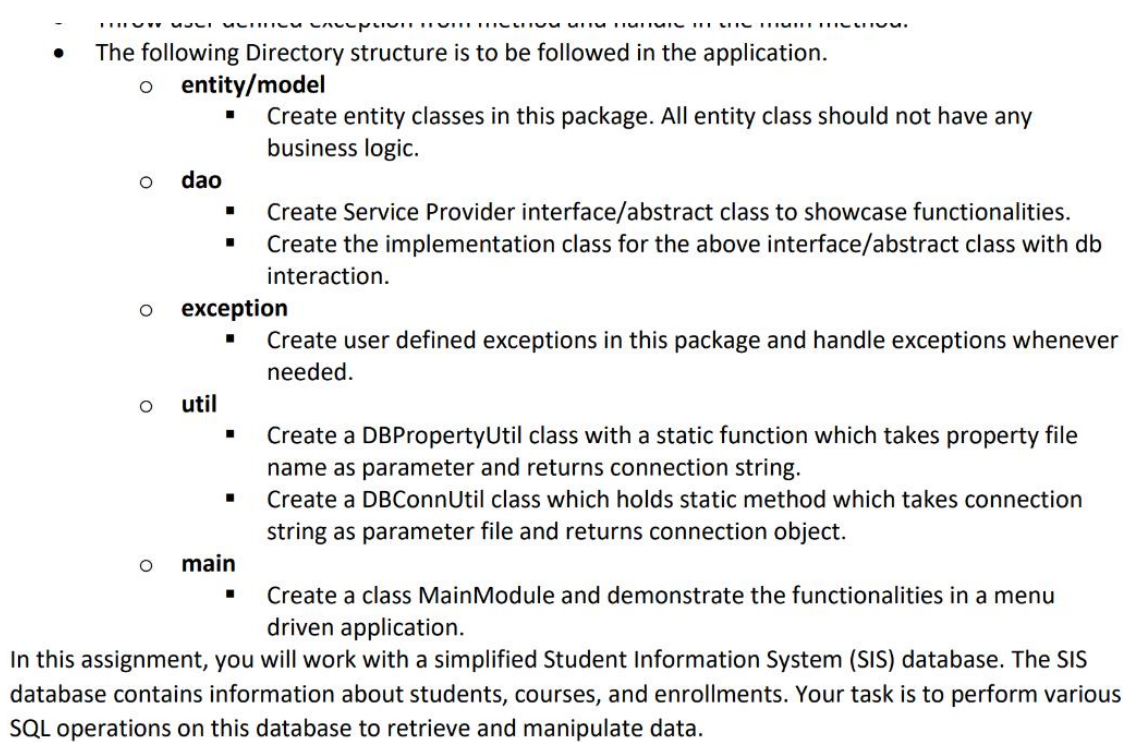
Student Information System (SIS)



- Throw user defined exception from method and handle in the main method.

- The following Directory structure is to be followed in the application.
  - **entity/model**
    - Create entity classes in this package. All entity class should not have any business logic.
  - **dao**
    - Create Service Provider interface/abstract class to showcase functionalities.
    - Create the implementation class for the above interface/abstract class with db interaction.
  - **exception**
    - Create user defined exceptions in this package and handle exceptions whenever needed.
  - **util**
    - Create a DBPropertyUtil class with a static function which takes property file name as parameter and returns connection string.
    - Create a DBConnUtil class which holds static method which takes connection string as parameter file and returns connection object.
  - **main**
    - Create a class MainModule and demonstrate the functionalities in a menu driven application.

In this assignment, you will work with a simplified Student Information System (SIS) database. The SIS database contains information about students, courses, and enrollments. Your task is to perform various SQL operations on this database to retrieve and manipulate data.

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

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

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

Payment class with the following attributes:

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

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.

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.

Enrollment Class:

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

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.

Payment Class:

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

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.

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.