Student Information System (SIS)

-V Chandana Priya Reddy

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 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 Student Information System (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 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

#Student class

```
class Student:
    def __init__(self, studentId,firstName, lastName, email, phoneNumber):
        self.studentId = studentId
        self.firstName = firstName
        self.lastName = lastName
        self.email = email
        self.phoneNumber = phoneNumber

    def set_studentId(self, studentId):
        self.studentId = studentId

    def set_firstName(self, firstName):
        self.firstName = firstName

    def set_lastName(self, lastName):
        self.lastName = lastName

    def set_email(self, email):
        self.email = email

    def set_phoneNumber(self, phoneNumber):
        self.phoneNumber = phoneNumber

    def get_studentId(self):
        return self.studentId
```

```
    def get_firstName(self):
        return self.firstName

    def get_lastName(self):
        return self.lastName

    def get_email(self):
        return self.email

    def get_phoneNumber(self):
        return self.phoneNumber


class Course:
    def __init__(self, courseId,courseName,courseCode, instructorName):
        self.courseId = courseId
        self.courseName = courseName
        self.courseCode = courseCode
        self.instructorName=instructorName

    def set_courseId(self,courseId):
        self.courseId = courseId
    def set_courseName(self,courseName):
        self.courseName = courseName
    def set_courseCode(self,courseCode):
        self.courseCode = courseCode
    def set_instructorName(self,instructorName):
        self.instructorName = instructorName

    def get_courseId(self):
        return self.courseId
    def get_courseName(self):
        return self.courseName
    def get_courseCode(self):
        return self.courseCode
    def get_instructorName(self):
        return self.instructorName


class Teacher:
    def __init__(self,teacherId,firstName,lastName,email):
        self.teacherId = teacherId
        self.firstName = firstName
        self.lastName = lastName
```

```
        self.email = email

    def set_teacherId(self,teacherId):
        self.teacherId = teacherId
    def set_firstName(self,firstName):
        self.firstName = firstName
    def set_lastName(self,lastName):
        self.lastName = lastName
    def set_email(self,email):
        self.email = email

    def get_teacherId(self):
        return self.teacherId
    def get_firstName(self):
        return self.firstName
    def get_lastName(self):
        return self.lastName
    def get_email(self):
        return self.email


class Enrollment:
    def __init__(self,enrollmentId,studentId,courseId,enrollmentDate):
        self.enrollmentId = enrollmentId
        self.studentId = studentId
        self.courseId = courseId
        self.enrollmentDate = enrollmentDate

    def set_enrollmentId(self,enrollmentId):
        self.enrollmentId = enrollmentId
    def set_studentId(self,studentId):
        self.studentId = studentId
    def set_courseId(self,courseId):
        self.courseId = courseId
    def set_enrollmentDate(self,enrollmentDate):
        self.enrollmentDate = enrollmentDate

    def get_enrollmentId(self):
        return self.enrollmentId
    def get_studentId(self):
        return self.studentId
    def get_courseId(self):
        return self.courseId
    def get_enrollmentDate(self):
        return self.enrollmentDate
```

```
class Payment:
    def __init__(self, paymentId,studentId,amount, paymentDate):
        self.paymentId = paymentId
        self.studentId = studentId
        self.amount = amount
        self.paymentDate = paymentDate

    def set_paymentId(self, paymentId):
        self.paymentId = paymentId
    def set_studentId(self, studentId):
        self.studentId = studentId
    def set_amount(self, amount):
        self.amount = amount
    def set_paymentDate(self, paymentDate):
        self.paymentDate = paymentDate

    def get_paymentId(self):
        return self.paymentId
    def get_studentId(self):
        return self.studentId
    def get_amount(self):
        return self.amount
    def get_paymentDate(self):
        return self.paymentDate
```

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 methods.

```
from abc import ABC,abstractmethod
class SIS(ABC):

    @abstractmethod
    def create_student(self):
        pass

    @abstractmethod
    def create_course(self):
        pass
```

```
@abstractmethod
def create_enrollment(self):
    pass

@abstractmethod
def create_teacher(self):
    pass

@abstractmethod
def create_payment(self):
    pass

@abstractmethod
def get_student_by_id(self):
    pass

@abstractmethod
def get_course_by_id(self):
    pass

@abstractmethod
def get_enrollments_for_student(self):
    pass

@abstractmethod
def get_enrollments_for_course(self):
    pass

@abstractmethod
def enroll_in_course(self):
    pass

@abstractmethod
def get_payments_for_student(self):
    pass

@abstractmethod
def get_payment_amount(self):
    pass

@abstractmethod
def get_payment_date(self):
    pass
```

```
@abstractmethod
def get_teacher_by_id(self):
    pass

@abstractmethod
def update_student(self):
    pass

@abstractmethod
def update_enrollment(self):
    pass

@abstractmethod
def update_payment(self):
    pass

@abstractmethod
def update_course(self):
    pass

@abstractmethod
def update_teacher(self):
    pass

@abstractmethod
def delete_student(self):
    pass

@abstractmethod
def delete_enrollment(self):
    pass

@abstractmethod
def delete_payment(self):
    pass

@abstractmethod
def delete_teacher(self):
    pass

@abstractmethod
def delete_course(self):
    pass

@abstractmethod
```

```
    def make_payment(self):
        pass

    @abstractmethod
    def get_enrolled_courses(self):
        pass

    @abstractmethod
    def assign_teacher(self, teacher):
        pass

    @abstractmethod
    def get_assigned_courses(self):
        pass

    @abstractmethod
    def get_courses_assigned_teacher(self):
        pass

    @abstractmethod
    def get_assigned_teacher_for_course(self):
        pass


from mycollections.mycollections import *
from customexceptions.myexception import *
from mycollections.mycollections import Student
from util.DBConnection import DBConnection


class StudentInformationImpl():
    def __init__(self):
        conn = DBConnection.getConnection()
        cursor = conn.cursor()

    def create_student(self):
        conn = DBConnection.getConnection()
        cursor = conn.cursor()
        try:
            self.firstName = input("Enter student first name: ")
            self.lastName = input("Enter student last name: ")
            self.dateOfBirth = input("Enter student date of birth: ")
            self.email = input("Enter student email id: ")
            self.phoneNumber = input("Enter student phone number: ")
            cursor.execute(
```

```
            "INSERT INTO Student (firstName, lastName, dateOfBirth, email, phoneNumber) "
            "VALUES (?,?,?,?,?)",
            (self.firstName, self.lastName, self.dateOfBirth, self.email, self.phoneNumber))
            print("Student created successfully")
            conn.commit()
            conn.close()
        except InvalidStudentDataException as e:
            print(e)
        except Exception as e:
            (print(str(e) + "---Error in creating student:---"))
        return None

    def create_course(self):
        conn = DBConnection.getConnection()
        cursor = conn.cursor()
        try:
            self.courseName= input("Enter course_name: ")
            self.courseCode = input("Enter course_code ")
            self.teacherId = int(input("Enter teacher id: "))
            cursor.execute(
            "INSERT INTO Course (courseName, courseCode, teacherId) VALUES (?,?,?)",
            (self.courseName, self.courseCode, self.teacherId))
            print("Course created successfully")
            conn.commit()
            conn.close()

        except InvalidCourseDataException as e:
                print(e)

        except Exception as e:
                (print(str(e) + "---error in creating course:---"))


    def create_enrollment(self):
        conn = DBConnection.getConnection()
        cursor = conn.cursor()
        try:
            self.student_id= int(input("Enter student_id: "))
            self.course_id= int(input("Enter course_id "))
            self.enrollment_date = input("Enter enrollment_date: ")
            cursor.execute(
                "INSERT INTO Enrollment (studentId, courseId, enrollmentDate) VALUES (?,?,?)",
                (self.student_id, self.course_id,self.enrollment_date))
            print("Enrollment created successfully")
            conn.commit()
```

```
            conn.close()
        except InvalidEnrollmentDataException as e:
            print(e)
        except Exception as e:
            (print(str(e) + "---error in creating enrollment:---"))

    def create_teacher(self):
        conn = DBConnection.getConnection()
        cursor = conn.cursor()
        try:
            self.firstName = input("Enter first_name: ")
            self.lastName = input("Enter last_name ")
            self.email = input("Enter email: ")

            cursor.execute(
                "INSERT INTO Teacher (firstName, lastName, email) VALUES (?,?,?)",
                (self.firstName, self.lastName, self.email))
            print("New Teacher has been created")

            conn.commit()
        except InvalidTeacherDataException as e:
            print(e)
        except Exception as e:
            (print(str(e) + "---error in creating teacher:---"))
        conn.close()
        return None

    def create_payment(self):
        conn = DBConnection.getConnection()
        cursor = conn.cursor()
        try:
            self.studentId = int(input("Enter student_id: "))
            self.amount = int(input("Enter amount "))
            self.paymentDate = input("Enter payment_date: ")

            cursor.execute("INSERT INTO Payment (studentId, amount, paymentDate) VALUES (?,?,?)",
                           (self.studentId, self.amount,self.paymentDate))
            print("New Payment has been created")

            conn.commit()
            conn.close()
        except InsufficientFundsException as e:
            print(e)
```

```
def get_student_by_id(self):
    conn = DBConnection.getConnection()
    cursor = conn.cursor()
    try:
        self.studentId = int(input("Enter student_id: "))
        cursor.execute("SELECT * FROM Student WHERE studentId = ?", (self.studentId,))
        student = [list(row) for row in cursor.fetchall()]
        if student:
            print(student)
        else:
            raise StudentNotFoundException("entered student_id is not found")
    except StudentNotFoundException as e:
            print(e)
    except Exception as e:(
            print(str(e) + "---error in getting students:---"))

def get_course_by_id(self):
    conn = DBConnection.getConnection()
    cursor = conn.cursor()
    try:
        self.courseId = int(input("Enter course_id: "))
        cursor.execute("SELECT * FROM Course WHERE courseId = ?",[(self.courseId)])
        row = cursor.fetchone()

        if row:
            course = Course(row[0], row[1], row[2], row[3])
            conn.close()
            print(course)
        else:
            raise CourseNotFoundException("entered course_id is not found")
    except CourseNotFoundException as e:
            print(e)


def get_enrollments_for_student(self):
    try:
        conn = DBConnection.getConnection()
        cursor = conn.cursor()
        self.student_id = int(input("Enter student_id: "))
        cursor.execute("SELECT * FROM Enrollment WHERE studentId = ?", [(self.student_id)])
        rows = cursor.fetchall()

        enrollments = []
        for row in rows:
            course = StudentInformationImpl.get_course_by_id(row[2])
```

```
            enrollment = Enrollment(row[0], Student, course, row[3])
            enrollments.append(enrollment)

            conn.close()
            print(enrollments)
        except InvalidEnrollmentDataException as e:
            print(e)

    def get_enrollments_for_course(self):
        try:
            conn = DBConnection.getConnection()
            cursor = conn.cursor()
            self.courseName = input("Enter courseName: ")
            cursor.execute(" SELECT s.firstName, s.lastName, e.enrollmentDate FROM Enrollment e JOIN
Student s ON e.studentId = s.studentId Join Course c on e.courseId=c.courseId WHERE courseName =
?",(self.courseName,))
            rows = cursor.fetchall()
            print(rows)

            return rows

        except InvalidEnrollmentDataException as e:
            print(e)

    def generate_report(self,courseName):
        courseName = input("Enter courseName: ")
        enrollments = self.get_enrollments_for_course(courseName)
        report = f"Enrollment Report for {courseName}:\n"
        for enrollment in enrollments:
            report += f"{enrollment[0]} {enrollment[1]} - {enrollment[2]}\n"
        print(report)

    def enroll_in_course(self):
        try:
            conn = DBConnection.getConnection()
            cursor = conn.cursor()
            self.student_id=int(input("enter student_id:"))
            self.course_id=int(input("course_id:"))
            self.enrollment_date=input("enter enrollment_date: ")
            cursor.execute( "INSERT into Enrollment (studentId,courseId,enrollmentDate) values (?,?,?)",
                    (self.student_id, self.course_id,self. enrollment_date))

            print("Enrollment has been created")
            conn.commit()
            cursor.close()
```

```
            conn.close()
            return True
        except InvalidCourseDataException as e:
            print(e)


    def get_payments_for_student(self):
        try:
            conn = DBConnection.getConnection()
            cursor = conn.cursor()
            self.student_id = int(input("Enter student_id: "))
            cursor.execute("SELECT * FROM Payment WHERE studentId = ?",[(self.student_id)])
            records = cursor.fetchall()
            if records:
                print('_______Records In payment Table___')
            for i in records:
                print(i)
            else:
                conn.close()
        except PaymentValidationException as e:
            print(e)


    def get_payment_amount(self):
        try:
            conn = DBConnection.getConnection()
            cursor = conn.cursor()
            self.student_id=int(input("enter student_id :"))
            cursor.execute("SELECT amount FROM Payment WHERE studentId = ?",[(self.student_id)])
            records = cursor.fetchall()
            if records:
                print('_______Records In payment Table___')
            for i in records:
                print(i)
            else:
                conn.close()
        except PaymentValidationException as e:
            print(e)


    def get_payment_date(self):
        try:
            conn = DBConnection.getConnection()
            cursor = conn.cursor()
            self.student_id=int(input("enter student_id :"))
```

```
            cursor.execute("SELECT paymentDate  FROM Payment WHERE studentId =
?",[(self.student_id)])
            records = cursor.fetchall()
            if records:
                print('_______Records In payment Table___')
            for i in records:
                print(i)
            else:
                conn.close()
        except PaymentValidationException as e:
            print(e)

    def get_teacher_by_id(self):
        try:
            conn = DBConnection.getConnection()
            cursor = conn.cursor()
            self.teacher_id = int(input("Enter teacher_id: "))
            cursor.execute("SELECT * FROM Teacher WHERE teacherId = ?",[(self.teacher_id )])
            records = cursor.fetchall()
            if records:
                print('_______Records In teacher Table___')
            for i in records:
                print(i)
            else:
                conn.close()
                raise InvalidTeacherDataException("entered teacher_id is not found")
        except TeacherNotFoundException as e:
            print(e)


    def update_student(self):
        try:
            conn = DBConnection.getConnection()
            cursor = conn.cursor()
            self.student_id = int(input("Enter student id: "))
            self.first_name = input("Enter student first name: ")
            self.last_name = input("Enter student last name: ")
            self.date_of_birth = input("Enter student date of birth: ")
            self.email = input("Enter student email id: ")
            self.phone_number = input("Enter student phone number: ")
            cursor.execute(
                "UPDATE Student SET firstName = ?, lastName = ?, dateOfBirth = ?, email = ?,
phoneNumber = ? WHERE studentId = ?",
                (self.first_name, self.last_name, self.date_of_birth, self.email,
```

```
self.phone_number,self.student_id))
            print("Student has been updated")

            conn.commit()
            conn.close()
            return True
        except InvalidStudentDataException as e:
            print(e)

    def update_enrollment(self):
        try:
            conn = DBConnection.getConnection()
            cursor = conn.cursor()
            self.enrollment_id = int(input("enrollment_id: "))
            self.student_id = int(input("Enter student_id: "))
            self.course_id = int(input("Enter course_id "))
            self.enrollment_date = input("Enter enrollment_date: ")

            cursor.execute(
                "UPDATE Enrollment SET  studentId=?, courseId=? , enrollmentDate=? WHERE enrollmentId
= ?",
                (self.student_id,self.course_id,self.enrollment_date,self.enrollment_id))
            print("Enrollment has been updated")
            conn.commit()
            conn.close()
        except InvalidEnrollmentDataException as e:
            print(e)

    def update_payment(self):
        try:
            conn = DBConnection.getConnection()
            cursor = conn.cursor()
            self.payment_id = int(input("Enter payment_id : "))
            self.student_id = int(input("Enter student_id: "))
            self.amount = int(input("Enter amount "))
            self.payment_date = input("Enter payment_date: ")

            cursor.execute(
            "UPDATE Payment SET   studentId=?, amount=?, paymentDate=? WHERE paymentId = ?",
            (self.payment_id,self.student_id,self.amount,self.payment_date))
            print("Payment has been updated")
            conn.commit()
            conn.close()
        except InsufficientFundsException as e:
            print(e)
```

```
def update_course(self):
    try:
        conn = DBConnection.getConnection()
        cursor = conn.cursor()
        self.course_id = int(input("Enter course_id: "))
        self.course_name = input("Enter course_name: ")
        self.course_code = input("Enter course_code ")
        self.teacher_id = int(input("Enter teacher_id: "))

        cursor.execute(
            "UPDATE Course SET courseName = ?, courseCode = ?, teacherId=? WHERE courseId = ?",
            (self.course_name, self.course_code,self.teacher_id,self.course_id))
        print("Course has been updated")
        conn.commit()
        conn.close()
    except InvalidCourseDataException as e:
        print(e)

def update_teacher(self):
    try:
        conn = DBConnection.getConnection()
        cursor = conn.cursor()
        self.teacher_id = int(input("Enter teacher id to update:"))
        self.first_name = input("Enter first name:")
        self.last_name = input("Enter last name:")
        self.email = input("Enter email:")
        sql = "UPDATE Teacher SET firstName = ?, lastName = ?, email = ? WHERE teacherId = ? "
        values = [(self.first_name, self.last_name, self.email, self.teacher_id)]
        print("Teacher has been updated")
        cursor.executemany(sql, values)
        conn.commit()
        conn.close()
    except InvalidTeacherDataException as e:
        print(e)

def delete_student(self):
    try:
        conn = DBConnection.getConnection()
        cursor = conn.cursor()
        self.student_id = int(input("Enter student_id to delete"))
        sql = "delete from  Student  WHERE studentId = ?"
        values = [(self.student_id)]
```

```
            cursor.execute(sql, values)
            print("Student has been deleted")
            conn.commit()
            conn.close()
        except StudentNotFoundException as e:
            print(e)

    def delete_enrollment(self):
        try:
            conn = DBConnection.getConnection()
            cursor = conn.cursor()
            self.enrollment_id = int(input("Enter enrollment_id to delete"))
            sql = "delete from  Enrollment  WHERE enrollmentId = ?"
            values = [(self.enrollment_id)]

            cursor.execute(sql, values)
            print("Enrollment has been deleted")
            conn.commit()
            conn.close()
        except InvalidEnrollmentDataException as e:
            print(e)

    def delete_payment(self):
        try:
            conn = DBConnection.getConnection()
            cursor = conn.cursor()
            self.payment_id = int(input("Enter payment_id to delete"))
            sql = "delete from  Payment  WHERE paymentId = ?"
            values = [self.payment_id]

            cursor.execute(sql, values)
            print("Payment has been deleted")
            conn.commit()
            conn.close()
        except PaymentValidationException as e:
            print(e)

    def delete_teacher(self):
        try:
            conn = DBConnection.getConnection()
            cursor = conn.cursor()
            self.teacher_id = int(input("Enter teacher_id to delete"))
            sql = "delete from  Teacher  WHERE teacherId = ?"
            values = [self.teacher_id]
```

```
            cursor.execute(sql, values)
            print("Teacher has been deleted")
            conn.commit()
            conn.close()
        except TeacherNotFoundException as e:
            print(e)

    def delete_course(self):
        try:
            conn = DBConnection.getConnection()
            cursor = conn.cursor()
            self.course_id = int(input("Enter course_id to delete"))
            sql = "delete from  Course  WHERE courseId = %s "
            values = [self.course_id]

            cursor.execute(sql, values)
            print("Course has been deleted")
            conn.commit()
            conn.close()
        except CourseNotFoundException as e:
            print(e)

    def make_payment(self):
        try:
            conn = DBConnection.getConnection()
            cursor = conn.cursor()
            self.student_id = int(input("Enter student_id: "))
            self.amount = int(input("Enter amount "))
            self.payment_date = input("Enter payment_date: ")

            cursor.execute("INSERT INTO Payment (studentId, amount, paymentDate) VALUES (?,?,?,?)",
                           (self.student_id, self.amount,self.payment_date))
            print("Payment has been made")
            conn.commit()
            conn.close()
        except InsufficientFundsException as e:
            print(e)

    def get_enrolled_courses(self):
            try:
                conn = DBConnection.getConnection()
                cursor = conn.cursor()
                self.student_id = int(input("Enter student_id: "))
                cursor.execute("SELECT * FROM Students WHERE studentId = %s", [(self.student_id)])
                records = cursor.fetchall()
```

```
            if records:
                print('_______Records In course Table___')
            for i in records:
                print(i)
            else:
                conn.close()
        except StudentNotFoundException as e:
            print(e)
    def assign_teacher(self):
        try:
            conn = DBConnection.getConnection()
            cursor = conn.cursor()
            self.course_id = int(input("Enter course_id: "))
            self.teacher_id = int(input("Enter teacher id: "))

            cursor.execute(
                "update  Course set teacherId =? WHERE course_id = ? ",(self.teacher_id,self.course_id))
            print("Teacher has been updated")
            conn.commit()
            conn.close()
        except CourseNotFoundException as e:
            print(e)
    def get_courses_assigned_teacher(self):
        try:
            conn = DBConnection.getConnection()
            cursor = conn.cursor()
            self.teacher_id= input("Enter teacher_id: ")

            cursor.execute(
                "select courseId,courseName from Course where teacherId=?", [(self.teacher_id)])
            conn.commit()
            conn.close()
        except  TeacherNotFoundException as e:
            print(e)

    def get_assigned_teacher_for_course(self):
        try:
            conn = DBConnection.getConnection()
            cursor = conn.cursor()
            self.course_id = int(input("Enter course_id: "))

            cursor.execute(
                "select teacherId from Course where courseId= ?",[(self.courseId)])
            conn.commit()
            conn.close()
```

```
    except TeacherNotFoundException as e:
        print(e)
```

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.

```
class DuplicateEnrollmentException(Exception):
    def __init__(self, message="Student is already enrolled in the course"):
        super().__init__(message)
        self.message = message

    def __str__(self):
        return self.message


class CourseNotFoundException(Exception):
```

```
    def __init__(self, message=" Course is not found"):
        super().__init__(message)
        self.message = message

    def __str__(self):
        return self.message


class StudentNotFoundException(Exception):
    def __init__(self, message="Student  not found"):
        super().__init__(message)
        self.message = message

    def __str__(self):
        return self.message


class TeacherNotFoundException(Exception):
    def __init__(self, message="Teacher not found"):
        super().__init__(message)
        self.message = message

    def __str__(self):
        return self.message


class PaymentValidationException(Exception):
    def __init__(self, message="Error in payment validation"):
        super().__init__(message)
        self.message = message

    def __str__(self):
        return self.message


class InvalidStudentDataException(Exception):
    def __init__(self, message="Invalid student data"):
        super().__init__(message)
        self.message = message

    def __str__(self):
        return self.message


class InvalidCourseDataException(Exception):
```

```
    def __init__(self, message="Invalid course data"):
        super().__init__(message)
        self.message = message

    def __str__(self):
        return self.message


class InvalidEnrollmentDataException(Exception):
    def __init__(self, message="Invalid enrollment data"):
        super().__init__(message)
        self.message = message

    def __str__(self):
        return self.message


class InvalidTeacherDataException(Exception):
    def __init__(self, message="Invalid teacher data"):
        super().__init__(message)
        self.message = message

    def __str__(self):
        return self.message


class InsufficientFundsException(Exception):
    def __init__(self, message="Insufficient funds for operation"):
        super().__init__(message)
        self.message = message

    def __str__(self):
        return self.message
```

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 accurate.

```
class Student:
    def _init_(self, student_id, first_name, last_name, dob, email, phone_number):
        self.student_id = student_id
        self.first_name = first_name
        self.last_name = last_name
```

```
        self.dob = dob
        self.email = email
        self.phone_number = phone_number
        self.enrollments = []  # List to hold enrollment objects

    def add_enrollment(self, enrollment):
        self.enrollments.append(enrollment)


class Course:
    def _init_(self, course_id, course_name="", course_code="", instructor_name=""):
        self.course_id = course_id
        self.course_name = course_name
        self.course_code = course_code
        self.instructor_name = instructor_name
        self.enrollments = []  # List to hold enrollments

    def add_enrollment(self, enrollment):
        self.enrollments.append(enrollment)


class Enrollment:
    def _init_(self, enrollment_id, student_id, course_id, enrollment_date):
        self.enrollment_id = enrollment_id
        self.student_id = student_id
        self.course_id = course_id
        self.enrollment_date = enrollment_date
        # Adding this enrollment to student and course upon creation
        student_id.add_enrollment(self)
        course_id.add_enrollment(self)


class Teacher:
    def _init_(self, teacher_id, first_name="", last_name="", email=""):
        self.teacher_id = teacher_id
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.assigned_courses = []  # List of courses assigned to this teacher

    def assign_course(self, course):
        self.assigned_courses.append(course)
        course.teacher = self
```

```
class Payment:
    def _init_(self, payment_id, student_id, amount, payment_date):
        self.payment_id = payment_id
        self.student_id = student_id
        self.amount = amount
        self.payment_date = payment_date
```

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

```
from dao.StudentInformationImpl import StudentInformationImpl

class MainModule:
    def __init__(self):
        self.service=StudentInformationImpl()

    def student_menu(self):
        while True:
            print("1.Create Student")
            print("2.Update Student")
            print("3.Delete Student")
            print("4.Display Student information")
            print("5.Student enrolled courses")
            print("6.Payments associated with Student")
            print("7.Return to Main Menu")

            option=input("Enter your choice: ")
            if option=="1":
                self.service.create_student()
```

```
        elif option=="2":
            self.service.update_student()
        elif option=="3":
            self.service.delete_student()
        elif option=="4":
            self.service.get_student_by_id()
        elif option=="5":
            self.service.get_enrolled_courses()
        elif option=="6":
            self.service.get_payments_for_student()
        elif option=="7":
            break
        else:
            print("Invalid option")

    def teacher_menu(self):
        while True:
            print("1.Create Teacher")
            print("2.Update Teacher")
            print("3.Delete Teacher")
            print("4.Display Teacher information")
            print("5.Teacher enrolled courses")
            print("6.Assign Teacher")
            print("7.Return to Main Menu")
            option=input("Enter your choice: ")
            if option=="1":
                self.service.create_teacher()
            elif option=="2":
                self.service.update_teacher()
            elif option=="3":
                self.service.delete_teacher()
            elif option=="4":
                self.service.get_teacher_by_id()
            elif option=="5":
                self.service.get_assigned_teacher_for_course()
            elif option=="6":
                self.service.assign_teacher()
            elif option=="7":
                break
            else:
                print("Invalid option")

    def course_menu(self):
        while True:
            print("1.Create Course")
```

```
            print("2.Update Course")
            print("3.Delete Course")
            print("4.Get Course by ID")
            print("5.Get assigned courses for the teacher")
            print("6.Return to Main Menu")
            option=input("Enter your choice: ")
            if option=="1":
                self.service.create_course()
            elif option=="2":
                self.service.update_course()
            elif option=="3":
                self.service.delete_course()
            elif option=="4":
                self.service.get_course_by_id()
            elif option=="5":
                self.service.get_courses_assigned_teacher()
            elif option=="6":
                break
            else:
                print("Invalid option")

    def enrollment_menu(self):
        while True:
            print("1.Create Enrollment")
            print("2.Update Enrollment")
            print("3.Delete Enrollment")
            print("4.Get Enrollments for Students")
            print("5.Get Enrollments for courses")
            print("6.Enroll in Course")
            print("7.Report")
            print("8.Return to Main Menu")
            option=input("Enter your choice: ")
            if option=="1":
                self.service.create_enrollment()
            elif option=="2":
                self.service.update_enrollment()
            elif option=="3":
                self.service.delete_enrollment()
            elif option=="4":
                self.service.get_enrollments_for_student()
            elif option=="5":
                self.service.get_enrollments_for_course()
            elif option=="6":
                self.service.enroll_in_course()
            elif option=="7":
```

```
            self.service.generate_report()
        elif option=="8":
            break
        else:
            print("Invalid option")

    def payment_menu(self):
        while True:
            print("1.Create Payment")
            print("2.Update Payment")
            print("3.Delete Payment")
            print("4.Make Payment")
            print("5.Get Payments for Students")
            print("6.Get Payment amount")
            print("7.Get payment Date")
            print("8.Return to Main Menu")
            option=input("Enter your choice: ")
            if option=="1":
                self.service.create_payment()
            elif option=="2":
                self.service.update_payment()
            elif option=="3":
                self.service.delete_payment()
            elif option=="4":
                self.service.make_payment()
            elif option=="5":
                self.service.get_payments_for_student()
            elif option=="6":
                self.service.get_payment_amount()
            elif option=="7":
                self.service.get_payment_date()
            elif option=="8":
                break
            else:
                print("Invalid option")



    def menu(self):
        menu=MainModule()
        while True:
            print("======Welcome to Student Information System======")
            print("Main Menu:")
            print("1.Students")
            print("2.Teachers")
```

```
        print("3.Courses")
        print("4.Enrollments")
        print("5.Payments")
        print("6.Exit")

        option=int(input("Enter your choice: "))
        if option==1:
            menu.student_menu()
        elif option==2:
            menu.teacher_menu()
        elif option==3:
            menu.course_menu()
        elif option==4:
            menu.enrollment_menu()
        elif option==5:
            menu.payment_menu()
        elif option==6:
            print("Thank you for using Student Information System")
            break
        else:
            print("Invalid option")


obj=MainModule()
obj.menu()
```

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.

```
import pyodbc
class DBPropertyUtil:
    @staticmethod
    def get_parameters():
        return {
            'Driver':'{SQL Server}',
            'host': 'Chandana\SQLEXPRESS',
            'database': 'SISDB',
            'user': 'Chandana',
            'password': 'Chandana03'
        }
```

```
class DBConnection:
    @staticmethod
    def getConnection():
        try:
            params = DBPropertyUtil.get_parameters()
            conn = pyodbc.connect(
                driver=params['Driver'],
                host=params['host'],
                database=params['database'],
                user=params['user'],
                password=params['password']
            )
            if conn:
                print('DB is connected: ')
            return conn
        except Exception as e:
            print(f"Error connecting to the database: {e}")
            return None
```

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

These all methods are implemented in dao and main modules.

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

```
======Welcome to Student Information System======
Main Menu:
1.Students
2.Teachers
3.Courses
4.Enrollments
5.Payments
6.Exit
Enter your choice: 1
1.Create Student
2.Update Student
3.Delete Student
4.Display Student information
5.Student enrolled courses
6.Payments associated with Student
7.Return to Main Menu
Enter your choice: 1
DB is connected:
Enter student first name: John
Enter student last name: Doe
Enter student date of birth: 1995-08-15
Enter student email id: john.doe@example.com
Enter student phone number: 1234567890
Student created successfully
```

Results Messages

	studentId	firstName	lastName	dateOfBirth	email	phoneNumber
1	1	Emily	Chen	1999-02-15	emilyc@gmail.com	1234567890
2	2	Ryan	Thompson	2001-10-22	ryan.t@gmail.com	9876543210
3	3	Priya	Singh	2000-08-12	psn@gmail.com	4567894321
4	4	Michael	Lee	1998-05-10	mike.lee@yahoo.com	7418529630
5	5	Sophia	Patel	2002-01-05	sophia.patel@hotmail.com	3692581470
6	6	David	Kim	1997-09-20	david.kim@outlook.com	8529637410
7	7	Olivia	Brown	2000-04-15	olivia.brown@gmail.com	1357924680
8	8	Ethan	Hall	1999-11-25	ethan.hall@yahoo.com	2468013579
9	9	Ava	Martin	2001-06-01	ava.martin@hotmail.com	9638527410
10	10	Liam	Davis	1998-03-10	liam.davis@outlook.com	5792468103
11	11	John	Doe	1995-08-15	john.doe@example.com	1234567890

John is enrolling in the following courses:

- Course 1: Introduction to Programming
- Course 2: Mathematics 101

```
======Welcome to Student Information System======
Main Menu:
1.Students
2.Teachers
3.Courses
4.Enrollments
5.Payments
6.Exit
Enter your choice: 4
1.Create Enrollment
2.Update Enrollment
3.Delete Enrollment
4.Get Enrollments for Students
5.Get Enrollments for courses
6.Enroll in Course
7.Return to Main Menu
Enter your choice: 6
DB is connected:
enter student_id:11
course_id:11
enter enrollment_date: 2024-10-11
Enrollment has been created
```

Results Messages

	enrollmentId	studentId	courseId	enrollmentDate
1	1	1	1	2022-08-15
2	2	2	2	2022-08-20
3	3	3	3	2022-08-25
4	4	4	4	2022-09-01
5	5	5	4	2022-09-05
6	6	6	6	2022-09-10
7	7	7	1	2022-09-12
8	8	8	8	2022-09-18
9	9	9	3	2022-09-22
10	10	10	10	2022-09-28
11	11	11	11	2024-10-11

```
======Welcome to Student Information System======
Main Menu:
1.Students
2.Teachers
3.Courses
4.Enrollments
5.Payments
6.Exit
Enter your choice: 4
1.Create Enrollment
2.Update Enrollment
3.Delete Enrollment
4.Get Enrollments for Students
5.Get Enrollments for courses
6.Enroll in Course
7.Return to Main Menu
Enter your choice: 6
DB is connected:
enter student_id:11
course_id:12
enter enrollment_date: 2024-10-11
Enrollment has been created
```

Results | Messages

	enrollmentId	studentId	courseId	enrollmentDate
1	1	1	1	2022-08-15
2	2	2	2	2022-08-20
3	3	3	3	2022-08-25
4	4	4	4	2022-09-01
5	5	5	4	2022-09-05
6	6	6	6	2022-09-10
7	7	7	1	2022-09-12
8	8	8	8	2022-09-18
9	9	9	3	2022-09-22
10	10	10	10	2022-09-28
11	11	11	11	2024-10-11
12	13	11	12	2024-10-11

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.

```
======Welcome to Student Information System======
Main Menu:
1.Students
2.Teachers
3.Courses
4.Enrollments
5.Payments
6.Exit
Enter your choice: 2
1.Create Teacher
2.Update Teacher
3.Delete Teacher
4.Display Teacher information
5.Teacher enrolled courses
6.Assign Teacher
7.Return to Main Menu
Enter your choice: 1
DB is connected:
Enter first_name: Sarah
Enter last_name Smith
Enter email: sarah.smith@example.com
New Teacher has been created
```

Results | Messages

	teacherId	firstName	lastName	email
1	1	Ram	Patel	ram@edu.in
2	2	Curie	Joe	joe@school.in
3	3	Sita	Singh	s@gmail.com
4	4	Rama	Martin	rama@edu.org
5	5	Raghu	Reddy	rr@email.com
6	6	Vishwa	Sen	sen@school.edu
7	7	Pavani	Rose	prose@school.org
8	8	Anitha	Rathode	anitha@gmail.com
9	9	Teju	Lee	lee@edu.in
10	10	Nani	Saheb	nani@org.in
11	11	Sarah	Smith	sarah.smith@example.com

```
======Welcome to Student Information System======
Main Menu:
1.Students
2.Teachers
3.Courses
4.Enrollments
5.Payments
6.Exit
Enter your choice: 3
1.Create Course
2.Update Course
3.Delete Course
4.Get Course by ID
5.Get assigned courses for the teacher
6.Return to Main Menu
Enter your choice: 1
DB is connected:
Enter course_name: Advanced Database Management
Enter course_code CS302
Enter teacher id: 11
Course created successfully
```

Results Messages

	courseId	courseName	courseCode	teacherId
2	2	Maths	M2	5
3	3	Computer Science	Cs2	11
4	4	C	C32	1
5	5	Python	P85	7
6	6	C++	C44	8
7	7	Java	J6	10
8	8	SQL	DB4	9
9	9	C#	C92	3
10	10	Skills	S1	4
11	11	Intoduction to Programming	P1	5
12	12	Mathematics	101	2
13	13	Advanced Database Management	CS302	11

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:

- 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.

```
======Welcome to Student Information System======
Main Menu:
1.Students
2.Teachers
3.Courses
4.Enrollments
5.Payments
6.Exit
Enter your choice: 1
1.Create Student
2.Update Student
3.Delete Student
4.Display Student information
5.Student enrolled courses
6.Payments associated with Student
7.Return to Main Menu
Enter your choice: 4
DB is connected:
Enter student_id: 12
[[12, 'jane', 'Johnson', '2000-10-10', 'john.s@gmail.com', 3456732456]]
```

Results Messages

	studentId	firstName	lastName	dateOfBirth	email	phoneNumber
1	1	Emily	Chen	1999-02-15	emilyc@gmail.com	1234567890
2	2	Ryan	Thompson	2001-10-22	ryan.t@gmail.com	9876543210
3	3	Priya	Singh	2000-08-12	psn@gmail.com	4567894321
4	4	Michael	Lee	1998-05-10	mike.lee@yahoo.com	7418529630
5	5	Sophia	Patel	2002-01-05	sophia.patel@hotmail.com	3692581470
6	6	David	Kim	1997-09-20	david.kim@outlook.com	8529637410
7	7	Olivia	Brown	2000-04-15	olivia.brown@gmail.com	1357924680
8	8	Ethan	Hall	1999-11-25	ethan.hall@yahoo.com	2468013579
9	9	Ava	Martin	2001-06-01	ava.martin@hotmail.com	9638527410
10	10	Liam	Davis	1998-03-10	liam.davis@outlook.com	5792468103
11	11	John	Doe	1995-08-15	john.doe@example.com	1234567890
12	12	jane	Johnson	2000-10-10	john.s@gmail.com	3456732456

```
======Welcome to Student Information System======
Main Menu:
1.Students
2.Teachers
3.Courses
4.Enrollments
5.Payments
6.Exit
Enter your choice: 5
1.Create Payment
2.Update Payment
3.Delete Payment
4.Make Payment
5.Get Payments for Students
6.Get Payment amount
7.Get payment Date
8.Return to Main Menu
Enter your choice: 1
DB is connected:
Enter student_id: 12
Enter amount 500
Enter payment_date: 2023-04-10
New Payment has been created
```

Results Messages

	paymentId	studentId	amount	paymentDate
1	1	1	1000	2022-08-15 00:00:00.000
2	2	2	500	2022-08-20 00:00:00.000
3	3	3	800	2022-08-25 00:00:00.000
4	4	4	1200	2022-09-01 00:00:00.000
5	5	5	900	2022-09-05 00:00:00.000
6	6	6	600	2022-09-10 00:00:00.000
7	7	7	1500	2022-09-12 00:00:00.000
8	8	8	700	2022-09-18 00:00:00.000
9	9	9	1100	2022-09-22 00:00:00.000
10	10	10	1300	2022-09-28 00:00:00.000
11	11	12	500	2023-04-10 00:00:00.000

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

```
======Welcome to Student Information System======
Main Menu:
1.Students
2.Teachers
3.Courses
4.Enrollments
5.Payments
6.Exit
Enter your choice: 4
1.Create Enrollment
2.Update Enrollment
3.Delete Enrollment
4.Get Enrollments for Students
5.Get Enrollments for courses
6.Enroll in Course
7.Report
8.Return to Main Menu
Enter your choice: 5
DB is connected:
Enter courseName: Computer Science
[('Priya', 'Singh', '2022-08-25'), ('Ava', 'Martin', '2022-09-22')]
```

```
======Welcome to Student Information System======
Main Menu:
1.Students
2.Teachers
3.Courses
4.Enrollments
5.Payments
6.Exit
Enter your choice: 4
1.Create Enrollment
2.Update Enrollment
3.Delete Enrollment
4.Get Enrollments for Students
5.Get Enrollments for courses
6.Enroll in Course
7.Report
8.Save the Report
9.Return to Main Menu
Enter your choice: 7
Enter courseName: Computer Science
DB is connected:
Enter courseName: Computer Science
Enrollment Report for Computer Science:
Priya Singh - 2022-08-25
Ava Martin - 2022-09-22
```