

Assignment 2

Task 1: Define Classes

Task 2: Implement Constructors

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

```
1  class Students:  
2      def __init__(self, student_id, first_name, last_name, dob, email, phone_number):  
3          self.studentID = student_id  
4          self.firstName = first_name  
5          self.lastName = last_name  
6          self.DOB = dob  
7          self.Email = email  
8          self.phoneNumber = phone_number  
9  |
```

Course class with the following attributes:

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

```
1  class Course:  
2      def __init__(self, CourseID, CourseName, CourseCode, InstructorName):  
3          self.CourseID = CourseID  
4          self.CourseName = CourseName  
5          self.CourseCode = CourseCode  
6          self.InstructorName = InstructorName  
7  |
```

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

```
1 class Enrollment:  
2     def __init__(self, enrollment_id, student, course, enrollment_date):  
3         self.EnrollmentID = enrollment_id  
4         self.Students = student  
5         self.Course = course  
6         self.EnrollmentDate = enrollment_date
```

Teacher class with the following attributes:

- Teacher ID
- First Name
- Last Name
- Email

```
1 class Teacher:  
2     def __init__(self, TeacherID, FirstName, LastName, Email):  
3         self.TeacherID = TeacherID  
4         self.FirstName = FirstName  
5         self.LastName = LastName  
6         self.Email = Email  
7
```

Payment class with the following attributes:

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

```
1 class Payment:  
2     def __init__(self, PaymentID, StudentID, Amount, PaymentDate):  
3         self.PaymentID = PaymentID  
4         self.StudentID = StudentID  
5         self.Amount = Amount  
6         self.PaymentDate = PaymentDate  
7
```

Task 3: Implement Methods

Task 5:

Student Class:

- EnrollInCourse(course: Course): Enrolls the student in a course.

```
 4 class Course:
 5     def __init__(self, course_id, course_name):
 6         self.courseID = course_id
 7         self.CourseName = course_name
 8
 9 class Students:
10     def __init__(self, student_id, first_name, last_name, dob, email, phone_number):
11         self.studentID = student_id
12         self.FirstName = first_name
13         self.lastName = last_name
14         self.DOB = dob
15         self.Email = email
16         self.phoneNumber = phone_number
17         self.EnrolledCourses = [] # List to store enrolled courses
18         self.PaymentHistory = [] # List to store payment records
19
20
21     def enroll_in_course(self, course):
22         self.EnrolledCourses.append(course)
23         print(f"Student {self.FirstName} enrolled in the course: {course.CourseName}")
24
25
26
27 s=Students(1, "John", "Doe", datetime(1990, 5, 15), "john.doe@example.com", "123-456-7890")
28 # s.update_student_info("John", "Doe", datetime(1990, 5, 15), "john.doe@example.com", "987-654-3210")
29
30 course1 = Course(101, "Introduction to Python")
31 course2 = Course(102, "Web Development Basics")
32 s.enroll_in_course(course1)
33 s.enroll_in_course(course2)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
print(f"Student {self.FirstName} enrolled in the course: {course.CourseName}")
```

attributeError: type object 'Course' has no attribute 'CourseName'

```
s: C:\Users\krish\OneDrive\Documents\Python> & C:/Python311/python.exe "c:/Users/krish/OneDrive/Documents/Python/Assignment1.py"
student John enrolled in the course: Introduction to Python
student John enrolled in the course: Web Development Basics
s: C:\Users\krish\OneDrive\Documents\Python> []
```

- UpdateStudentInfo(firstName: string, lastName: string, dateOfBirth: DateTime, email: string, phoneNumber: string): Updates the student's information.

```

18     def update_student_info(self, first_name, last_name, dob, email, phone_number):
19         self.FirstName = first_name
20         self.LastName = last_name
21         self.DateOfBirth = dob
22         self.Email = email
23         self.PhoneNumber = phone_number
24         print("Student information updated successfully.")
25
26
27 s=Students(1, "John", "Doe", datetime(1990, 5, 15), "john.doe@example.com", "123-456-7890")
28 s.update_student_info("John", "Doe", datetime(1990, 5, 15), "john.doe@example.com", "987-654-3210")
29
30

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\Users\krish\OneDrive\Documents\Python> & c:/Python311/python.exe "c:/users/krish/OneDrive/Documents/Python/Assignment 2/Assignment 2.py"
PS C:\Users\krish\OneDrive\Documents\Python> & c:/Python311/python.exe "c:/users/krish/OneDrive/Documents/Python/Assignment 2/Assignment 2.py"
Student information updated successfully.
PS C:\Users\krish\OneDrive\Documents\Python> []

```

- MakePayment(amount: decimal, paymentDate: DateTime): Records a payment made by the student.

```

37     def make_payment(self, amount, payment_date):
38         payment = Payment(amount, payment_date)
39         self.PaymentHistory.append(payment)
40         print(f"Payment of ${amount} recorded on {payment_date}")
41
42 s=Students(1, "John", "Doe", datetime(1990, 5, 15), "john.doe@example.com", "123-456-7890")
43 # s.update_student_info("John", "Doe", datetime(1990, 5, 15), "john.doe@example.com", "987-654-3210")
44
45 # course1 = Course(101, "Introduction to Python")
46 # course2 = Course(102, "Web Development Basics")
47 # s.enroll_in_course(course1)
48 # s.enroll_in_course(course2)
49
50 s.make_payment(50.0, datetime(2023, 1, 20))
51 s.make_payment(75.0, datetime(2023, 2, 15))

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

s=Students()
^^^^^^^^^
TypeError: Students.__init__() missing 6 required positional arguments: 'student_id', 'first_name', 'last_name', 'dob', 'email', 'phone_number'
PS C:\Users\krish\OneDrive\Documents\Python> & c:/Python311/python.exe "c:/users/krish/OneDrive/Documents/Python/Assignment 2/Assignment 2.py"
Payment of $50.0 recorded on 2023-01-20 00:00:00
Payment of $75.0 recorded on 2023-02-15 00:00:00
PS C:\Users\krish\OneDrive\Documents\Python> []

```

- DisplayStudentInfo(): Displays detailed information about the student.

```

1  from datetime import datetime
2
3  class Teacher:
4      def __init__(self, teacher_id, teacher_name):
5          self.TeacherID = teacher_id
6          self.TeacherName = teacher_name
7
8  class Enrollment:
9      def __init__(self, enrollment_id, student, course, enrollment_date):
10         self.EnrollmentID = enrollment_id
11         self.Student = student
12         self.Course = course
13         self.EnrollmentDate = enrollment_date
14
15  class Course:
16      def __init__(self, course_code, course_name, instructor):
17          self.CourseCode = course_code
18          self.CourseName = course_name
19          self.Instructor = instructor
20          self.AssignedTeacher = None # Initially, no teacher is assigned
21          self.Enrollments = [] # List to store student enrollments
22
23      def assign_teacher(self, teacher):
24          self.AssignedTeacher = teacher
25          print(f"Teacher {teacher.TeacherName} assigned to the course: {self.CourseName}")
26
27
28
29  # Example usage:
30 teacher = Teacher(1, "Dr. Smith")
31 course = Course("CS101", "Introduction to Computer Science", "Prof. Johnson")

```

PROBLEMS ② OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

Course.assign_teacher(teacher)
TypeError: Course.assign_teacher() missing 1 required positional argument: 'teacher'
PS C:\Users\krish\OneDrive\Documents\Python> & C:/Python311/python.exe "c:/Users/krish/OneDrive/Documents/Python"
Teacher Dr. Smith assigned to the course: Introduction to Computer Science

```

- UpdateCourseInfo(courseCode: string, courseName: string, instructor: string): Updates course information.

```

27      def update_course_info(self, course_code, course_name, instructor):
28          self.CourseCode = course_code
29          self.CourseName = course_name
30          self.Instructor = instructor
31          print("Course information updated successfully.")

```

PROBLEMS ② OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\Users\krish\OneDrive\Documents\Python> & C:/Python311/python.exe "c:/Users/krish/OneDrive/Do
Teacher Dr. Smith assigned to the course: Introduction to Computer Science
PS C:\Users\krish\OneDrive\Documents\Python> & C:/Python311/python.exe "c:/Users/krish/OneDrive/Do
Teacher Dr. Smith assigned to the course: Introduction to Computer Science
PS C:\Users\krish\OneDrive\Documents\Python> & C:/Python311/python.exe "c:/Users/krish/OneDrive/Do
Course information updated successfully.
PS C:\Users\krish\OneDrive\Documents\Python> []

```

- DisplayCourseInfo(): Displays detailed information about the course.

```

33     def display_course_info(self):
34         print(f"Course Code: {self.CourseCode}")
35         print(f"Course Name: {self.CourseName}")
36         print(f"Instructor: {self.Instructor}")
37         if self.AssignedTeacher:
38             print(f"Assigned Teacher: {self.AssignedTeacher.TeacherName}")

39
40     # def get_enrollments(self):
41     #     return self.Enrollments
42
43     # def get_teacher(self):
44     #     return self.AssignedTeacher
45
46     # Example usage:
47     #teacher = Teacher(1, "Dr. Smith")
48     course = Course("CS101", "Introduction to Computer Science", "Prof. Johnson")
49
50     # course.assign_teacher(teacher)
51     #course.update_course_info("CS102", "Data Structures", "Prof. Brown")
52
53     course.display_course_info()
54

```

PROBLEMS ② OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\Users\krish\OneDrive\Documents\Python> & C:/Python311/python.exe "c:/Users/krish/OneDrive/Documents\Python\CourseInformation.py"
Course information updated successfully.
PS C:\Users\krish\OneDrive\Documents\Python> & C:/Python311/python.exe "c:/Users/krish/OneDrive/Documents\Python\CourseInformation.py"
Course Code: CS101
Course Name: Introduction to Computer Science
Instructor: Prof. Johnson
PS C:\Users\krish\OneDrive\Documents\Python> []

```

- GetEnrollments(): Retrieves a list of student enrollments for the course.
- GetTeacher(): Retrieves the assigned teacher for the course.

```

40     def get_enrollments(self):
41         return self.Enrollments
42
43     def get_teacher(self):
44         return self.AssignedTeacher
45

```

Enrollment Class:

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

```
1 from datetime import datetime
2 from students import Student
3 from teacher import Teacher
4 from course import Course
5 from enrollment import Enrollment
6 # Create instances of Student, Teacher, and Course
7 # student = Student(1, "John Doe")
8 # teacher = Teacher(1, "Dr. Smith")
9 # course = Course("CS101", "Introduction to Computer Science", "Prof. Johnson")
10
11 # Assign the teacher to the course
12 # course.assign_teacher(teacher)
13
14 # Create an instance of Enrollment
15 enrollment = Enrollment(1, Student, Course, datetime.now())
16
17 # Retrieve and print the associated student and course
18 associated_student = enrollment.get_student()
19 associated_course = enrollment.get_course()
20
21 print("Student:", associated_student.FirstName) # Assuming FirstName is an attribute
22 print("course:", associated_course.CourseName) # Assuming CourseName is an attribute
23
```

PROBLEMS ② OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
print("Course:", associated_course.CourseName) # Assuming CourseName is an attribute in
                                                ^^^^^^^^^^
AttributeError: type object 'Course' has no attribute 'CourseName'
PS C:\Users\krish\OneDrive\Documents\Python> & C:/Python311/python.exe "c:/Users/krish/OneDrive/Students/Krishna.py"
Student: Krishna
Course: Introduction to Computer Science
PS C:\Users\krish\OneDrive\Documents\Python> []
```

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.

```

2  class Teacher:
3      def __init__(self, teacher_id, name):
4          self.TeacherID = teacher_id
5          self.Name = name
6          self.Email = ""
7          self.Expertise = ""
8          self.AssignedCourses = []
9
10     def update_teacher_info(self, name, email, expertise):
11         self.Name = name
12         self.Email = email
13         self.Expertise = expertise
14
15     def display_teacher_info(self):
16         print(f"Teacher ID: {self.TeacherID}")
17         print(f"Name: {self.Name}")
18         print(f"Email: {self.Email}")
19         print(f"Expertise: {self.Expertise}")
20         print("Assigned Courses:", ", ".join(course.CourseName for course in self.AssignedCourses))
21
22     def get_assigned_courses(self):
23         return self.Assignedcourses
24

```

PROBLEMS ② OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

TypeError: 'Course' object is not callable
PS C:\Users\krish\OneDrive\Documents\Python> & C:/Python311/python.exe "c:/Users/krish/OneDrive/Documents/Python/Ass
Teacher ID: 1
Name: Dr. Emily
Email: emily@example.com
Expertise: Data Science
Assigned Courses: Introduction to Computer Science, Advanced Algorithms
Assigned Courses: Introduction to Computer Science, Advanced Algorithms
PS C:\Users\krish\OneDrive\Documents\Python> []

```

Payment Class:

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

```
1 class Payment:
2     def __init__(self, payment_id, student, amount, payment_date):
3         self.PaymentID = payment_id
4         self.Student = student
5         self.Amount = amount
6         self.PaymentDate = payment_date
7
8     def get_student(self):
9         return self.Student
10
11    def get_payment_amount(self):
12        return self.Amount
13
14    def get_payment_date(self):
15        return self.PaymentDate
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\krish\OneDrive\Documents\Python> & C:/Python311/python.exe "c:/Users/krish/PycharmProjects/SIS/main.py"
Payment Details:
Student: John Doe
Amount: 500.0
Payment Date: 2023-12-01
PS C:\Users\krish\OneDrive\Documents\Python>
```

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.

```

2  from students import Student
3  from course import Course
4  from teacher import Teacher
5  from payment import Payment
6  from enrollment import Enrollment
7
8  class SIS:
9      def __init__(self):
10         self.enrollments = []
11         self.payments = []
12
13     def enroll_student_in_course(self, Student, Course):
14         enrollment = Enrollment(Student, Course)
15         self.enrollments.append(enrollment)
16
17     def assign_teacher_to_course(self, Teacher, Course):
18         Course.assign_teacher(Teacher)
19
20     def record_payment(self, student, amount, payment_date):
21         payment = Payment(student, amount, payment_date)
22         self.payments.append(payment)
23
24     def generate_enrollment_report(self, course):
25         enrolled_students = [enrollment.get_student() for enrollment in self.enrollments if enrollment.get_course() == course]
26         return enrolled_students
27
28     def generate_payment_report(self, student):
29         student_payments = [payment for payment in self.payments if payment.get_student() == student]
30         return student_payments
31
32     def calculate_course_statistics(self, course):
33         enrollments_count = len([enrollment for enrollment in self.enrollments if enrollment.get_course() == course])
34         total_payments = sum(payment.get_amount() for payment in self.payments if payment.get_student().get_courses() == course)
35         return {'enrollments_count': enrollments_count, 'total_payments': total_payments}

```

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

```
1  class DuplicateEnrollmentException(Exception):
2      def __init__(self, message="Student is already enrolled in the course."):
3          self.message = message
4          super().__init__(self.message)
5
6  class CourseNotFoundException(Exception):
7      def __init__(self, message="Course not found in the system."):
8          self.message = message
9          super().__init__(self.message)
10
11 class StudentNotFoundException(Exception):
12     def __init__(self, message="Student not found in the system."):
13         self.message = message
14         super().__init__(self.message)
15
16 class TeacherNotFoundException(Exception):
17     def __init__(self, message="Teacher not found in the system."):
18         self.message = message
19         super().__init__(self.message)
20
21 class PaymentValidationException(Exception):
22     def __init__(self, message="Payment validation failed."):
23         self.message = message
24         super().__init__(self.message)
25
26 class InvalidStudentDataException(Exception):
27     def __init__(self, message="Invalid data for creating or updating a student."):
28         self.message = message
29         super().__init__(self.message)
30
31 class InvalidCourseDataException(Exception):
32     def __init__(self, message="Invalid data for creating or updating a course."):
33         self.message = message
34         super().__init__(self.message)
```

```
36 class InvalidEnrollmentDataException(Exception):
37     def __init__(self, message="Invalid data for creating an enrollment."):
38         self.message = message
39         super().__init__(self.message)
40
41 class InvalidTeacherDataException(Exception):
42     def __init__(self, message="Invalid data for creating or updating a teacher."):
43         self.message = message
44         super().__init__(self.message)
45
46 class InsufficientFundsException(Exception):
47     def __init__(self, message="Insufficient funds to enroll in the course."):
48         self.message = message
49         super().__init__(self.message)
```

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.

```

102     def add_enrollment(self, student, course, enrollment_date):
103         # Check if the student and course exist in the system
104         if student not in self.students:
105             raise StudentNotFoundException("Student not found in the system.")
106
107         if course not in self.courses:
108             raise CourseNotFoundException("Course not found in the system.")
109
110         # Create an enrollment and add it to both the student's and course's enrollment lists
111         enrollment = Enrollment(student, course, enrollment_date)
112         student.enrollments.append(enrollment)
113         course.enrollments.append(enrollment)
114
115     def assign_course_to_teacher(self, course, teacher):
116         # Check if the course and teacher exist in the system
117         if course not in self.courses:
118             raise CourseNotFoundException("Course not found in the system.")
119
120         if teacher not in self.teachers:
121             raise TeacherNotFoundException("Teacher not found in the system.")
122
123         # Assign the course to the teacher
124         teacher.assign_course(course)
125
126     def add_payment(self, student, amount, payment_date):
127         # Check if the student exists in the system
128         if student not in self.students:
129             raise StudentNotFoundException("Student not found in the system.")
130
131         # Create a payment and add it to the student's payment history
132         payment = Payment(amount=amount, payment_date=payment_date, student=student)
133         student.PaymentHistory.append(payment)
134
135     def get_enrollments_for_student(self, student):
136         # Check if the student exists in the system
137         if student not in self.students:
138             raise StudentNotFoundException("Student not found in the system.")
139
140         # Retrieve all enrollments for the student
141         return student.enrollments
142
143     def get_courses_for_teacher(self, teacher):
144         # Check if the teacher exists in the system
145         if teacher not in self.teachers:
146             raise TeacherNotFoundException("Teacher not found in the system.")
147
148         # Retrieve all courses assigned to the teacher
149         return teacher.assigned_courses
150
151 sis = SIS()
152
153 # Create students, courses, and teachers
154 student1 = Student(1, "Krishna", "Patle", datetime(2001, 8, 12), "krishnapatle@128.com", "9325654953")
155 student2 = Student(2, "Jane", "Doe", datetime(2002, 9, 12), "johndoe@gmail.com", "9157483331")
156
157 course1 = Course(course_code="C001", course_name="Introduction to Python")
158 course2 = Course(course_code="C002", course_name="Data Structures")
159
160 teacher1 = Teacher(teacher_id=1, name="Prof. Smith")
161 teacher2 = Teacher(teacher_id=2, name="Prof. Johnson")
162
163 # Add students, courses, and teachers to the SIS
164 sis.students = [student1, student2]
165 sis.courses = [course1, course2]
166 sis.teachers = [teacher1, teacher2]
167

```

```

164 sis.students = [student1, student2]
165 sis.courses = [course1, course2]
166 sis.teachers = [teacher1, teacher2]
167
168 # Add enrollments, assign courses to teachers, and add payments
169 try:
170     sis.add_enrollment(student1, course1, enrollment_date="2023-01-01")
171     sis.add_enrollment(student1, course2, enrollment_date="2023-01-15") # Duplicate enrollment should
172
173     sis.assign_course_to_teacher(course1, teacher1)
174     sis.assign_course_to_teacher(course2, teacher2)
175
176     sis.add_payment(student1, amount=500, payment_date="2023-02-01")
177 except (DuplicateEnrollmentException, CourseNotFoundException, StudentNotFoundException,
178         TeacherNotFoundException, PaymentValidationException) as e:
179     print(f"Error: {str(e)}")
180
181 # Get enrollments for a student
182 enrollments_for_student1 = sis.get_enrollments_for_student(student1)
183 print("Enrollments for Student 1:")
184 for enrollment in enrollments_for_student1:
185     print(f"Course: {enrollment.course.course_name}, Enrollment Date: {enrollment.enrollment_date}")
186
187 # Get courses assigned to a teacher
188 courses_for_teacher1 = sis.get_courses_for_teacher(teacher1)
189 print("Courses assigned to Teacher 1:")
190 for course in courses_for_teacher1:
191     print(f"Course: {course.course_name}")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\Users\krish\OneDrive\Documents\Python> & C:/Python311/python.exe "c:/Users/krish/OneDrive/Documents/
Enrollments for Student 1:
Course: Introduction to Python, Enrollment Date: 2023-01-01
Course: Data Structures, Enrollment Date: 2023-01-15
Courses assigned to Teacher 1:
Course: Introduction to Python
PS C:\Users\krish\OneDrive\Documents\Python>

```

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.

```
59 import mysql.connector
60 class DBUtil:
61     def __init__(self, host, user, password, port, database):
62         self.connection = mysql.connector.connect(
63             host=host,
64             user=user,
65             password=password,
66             port=port,
67             database=database
68         )
69         self.cursor = self.connection.cursor()
70
71     def execute_query(self, query, values=None):
72         try:
73             self.cursor.execute(query, values)
74             if self.cursor.description is not None:
75                 self.cursor.fetchall()
76             else:
77                 self.connection.commit()
78         except Exception as e:
79             print(f"Error executing query: {str(e)}")
80             self.connection.rollback()
81
82     def fetch_one(self, query, values=None):
83         self.cursor.execute(query, values)
84         return self.cursor.fetchone()
85
86     def fetch_all(self, query, values=None):
87         self.cursor.execute(query, values)
88         return self.cursor.fetchall()
89
90     def close_connection(self):
91         self.cursor.close()
92         self.connection.close()
```

```
105 def initialize_database(db_util):
106     create_tables_query = """
107     CREATE TABLE IF NOT EXISTS students (
108         student_id INT PRIMARY KEY,
109         name VARCHAR(255)
110     );
111
112     CREATE TABLE IF NOT EXISTS courses (
113         course_code VARCHAR(10) PRIMARY KEY,
114         course_name VARCHAR(255)
115     );
116
117     CREATE TABLE IF NOT EXISTS enrollments (
118         enrollment_id INT PRIMARY KEY,
119         student_id INT,
120         course_code VARCHAR(10),
121         enrollment_date DATE,
122         FOREIGN KEY (student_id) REFERENCES students(student_id),
123         FOREIGN KEY (course_code) REFERENCES courses(course_code)
124     );
125
126     CREATE TABLE IF NOT EXISTS teachers (
127         teacher_id INT PRIMARY KEY,
128         name VARCHAR(255)
129     );
130
131     CREATE TABLE IF NOT EXISTS payments (
132         payment_id INT PRIMARY KEY,
133         student_id INT,
134         amount DECIMAL(10, 2),
135         payment_date DATE,
136         FOREIGN KEY (student_id) REFERENCES students(student_id)
137     );
138     """
139
    db_util.execute_query(create_tables_query)
```

```
142     def get_students(db_util):
143         query = "SELECT * FROM students"
144         return db_util.fetch_all(query)
145
146     def get_courses(db_util):
147         query = "SELECT * FROM courses"
148         return db_util.fetch_all(query)
149
150     # Data Insertion
151     def insert_student(db_util, student_id, name):
152         query = "INSERT INTO students (student_id, name) VALUES (%s, %s)"
153         values = (student_id, name)
154         db_util.execute_query(query, values)
155
156     # Transaction Management
157     def enroll_student(db_util, student_id, course_code, enrollment_date):
158         try:
159             db_util.connection.start_transaction()
160
161             # check if student and course exist
162             student_query = "SELECT * FROM students WHERE student_id = %s"
163             course_query = "SELECT * FROM courses WHERE course_code = %s"
164             student = db_util.fetch_one(student_query, (student_id,))
165             course = db_util.fetch_one(course_query, (course_code,))
166
167             if not student or not course:
168                 raise Exception("Student or course not found")
169
170             # Enroll the student
171             enrollment_query = "INSERT INTO enrollments (student_id, course_code, enrollment_date) VALUES (%s, %s, %s)"
172             enrollment_values = (student_id, course_code, enrollment_date)
173             db_util.execute_query(enrollment_query, enrollment_values)
174
175             db_util.connection.commit()
176         except Exception as e:
177             print(f"Error enrolling student: {str(e)}")
178             db_util.connection.rollback()
179         finally:
180             db_util.connection.autocommit = True
```

```

182 # Dynamic Query Builder
183 def execute_custom_query(db_util, query, values=None):
184     return db_util.fetch_all(query, values)
185
186 # Example usage
187 db_util = DBUtil(host='localhost', user='root', password='Krishna@128', port="3306", database='sis')
188 initialize_database(db_util)
189
190 # Insert a student
191 insert_student(db_util, student_id=1, name="John Doe")
192
193 # Enroll a student in a course
194 enroll_student(db_util, student_id=1, course_code="C001", enrollment_date="2023-01-01")
195
196 # Get students and courses
197 students = get_students(db_util)
198 courses = get_courses(db_util)
199
200 print("Students:")
201 print(students)
202
203 print("Courses:")
204 print(courses)
205
206 # Execute a custom query
207 custom_query = "SELECT * FROM enrollments WHERE student_id = %s"
208 custom_query_values = (1,)
209 enrollments = execute_custom_query(db_util, custom_query, custom_query_values)
210
211 print("Enrollments:")
212 print(enrollments)
213
214 # Close the database connection
215 db_util.close_connection()
216

```

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.

```
218 import mysql.connector
219 from datetime import date, datetime
220
221 # Database connection
222 db_conn = mysql.connector.connect(
223     host="localhost",
224     user="root",
225     password="krishna@128",
226     database="sis"
227 )
228 cursor = db_conn.cursor()
229
230 class Student:
231     def __init__(self, first_name, last_name, date_of_birth, email, phone_number):
232         self.first_name = first_name
233         self.last_name = last_name
234         self.date_of_birth = date_of_birth
235         self.email = email
236         self.phone_number = phone_number
237
238     def save_to_database(self):
239         query = "INSERT INTO students (first_name, last_name, date_of_birth, email, phone_number) VALUES (%s, %s, %s, %s, %s)"
240         values = (self.first_name, self.last_name, self.date_of_birth, self.email, self.phone_number)
241         cursor.execute(query, values)
242         db_conn.commit()
243         print("Student information saved to the database.")
244
245 class Course:
246     def __init__(self, course_code, course_name):
247         self.course_code = course_code
248         self.course_name = course_name
249
250 class Enrollment:
251     def __init__(self, student_id, course_code):
252         self.student_id = student_id
253         self.course_code = course_code
```

```

255     def save_to_database(self):
256         query = "INSERT INTO enrollments (student_id, course_code) VALUES (%s, %s)"
257         values = (self.student_id, self.course_code)
258         cursor.execute(query, values)
259         db_conn.commit()
260         print("Enrollment record saved to the database.")
261
262     cursor.execute('''
263         CREATE TABLE IF NOT EXISTS students (
264             student_id INT AUTO_INCREMENT PRIMARY KEY,
265             first_name VARCHAR(255) NOT NULL,
266             last_name VARCHAR(255) NOT NULL,
267             date_of_birth DATE,
268             email VARCHAR(255),
269             phone_number VARCHAR(20)
270         )
271     ''')
272
273     cursor.execute('''
274         CREATE TABLE IF NOT EXISTS enrollments (
275             enrollment_id INT AUTO_INCREMENT PRIMARY KEY,
276             student_id INT,
277             course_code VARCHAR(50),
278             FOREIGN KEY (student_id) REFERENCES students(student_id),
279             FOREIGN KEY (course_code) REFERENCES courses(course_code)
280         )
281     ''')
282
283     john_doe = student(
284         first_name='John',
285         last_name='Doe',
286         date_of_birth=date(1995, 8, 15),
287         email='john.doe@example.com',
288         phone_number='123-456-7890'
289     )
290     john_doe.save_to_database()
291
292     course_1 = Course(course_code='COURSE1', course_name='Introduction to Programming')
293     course_2 = Course(course_code='COURSE2', course_name='Mathematics 101')
294
295     enrollment_1 = Enrollment(student_id=1, course_code='COURSE1') # Assuming John Doe's ID is 1
296     enrollment_1.save_to_database()
297
298     enrollment_2 = Enrollment(student_id=1, course_code='COURSE2')
299     enrollment_2.save_to_database()
300
301     # Close Database Connection
302     cursor.close()
303     db_conn.close()
304

```

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.

```
1 import mysql.connector
2
3 db_conn = mysql.connector.connect(
4     host="localhost",
5     user="root",
6     password='krishna@128',
7     database='sis'
8 )
9 cursor = db_conn.cursor()
10
11 teacher_name = 'Sarah Smith'
12 teacher_email = 'sarah.smith@example.com'
13 teacher_expertise = 'Computer Science'
14
15 course_code = 'CS302'
16 new_instructor_name = 'Sarah Smith'
17
18 cursor.execute("SELECT * FROM courses WHERE course_code = %s", (course_code,))
19 course_record = cursor.fetchone()
20
21 if course_record:
22     cursor.execute("UPDATE courses SET instructor = %s WHERE course_code = %s", (new_instructor_name, course_code))
23     db_conn.commit()
24     print(f"[teacher_name] assigned as the instructor for the course {course_code}.")
25 else:
26     print(f"Course with code {course_code} not found.")
27
28 cursor.close()
29 db_conn.close()
30
31
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\krish\OneDrive\Documents\Python> & C:/Python311/python.exe "c:/Users/krish/OneDrive/Documents/Python/Assignment 2/course.py"
Course with code CS302 not found.
PS C:\Users\krish\OneDrive\Documents\Python>

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.

```
1 import mysql.connector
2 from datetime import date
3
4 db_conn = mysql.connector.connect(
5     host="localhost",
6     user="root",
7     password="Krishna@128",
8     database="sis"
9 )
10 cursor = db_conn.cursor()
11
12 student_id = 101
13 payment_amount = 500.00
14 payment_date = date(2023, 4, 10)
15
16 cursor.execute("SELECT * FROM students WHERE student_id = %s", (student_id,))
17 student_record = cursor.fetchone()
18
19 if student_record:
20     cursor.execute("INSERT INTO payments (student_id, amount, payment_date) VALUES (%s, %s, %s)",
21                   (student_id, payment_amount, payment_date))
22     db_conn.commit()
23
24     cursor.execute("UPDATE students SET outstanding_balance = outstanding_balance - %s WHERE student_id = %s",
25                   (payment_amount, student_id))
26     db_conn.commit()
27
28     print(f"Payment recorded for student {student_record[1]} {student_record[2]} (Student ID: {student_id}).")
29 else:
30     print(f"Student with ID {student_id} not found.")
31
32 cursor.close()
33 db_conn.close()
34
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Course with code CS302 not found.
PS C:\Users\Varish\OneDrive\Documents\Python> & C:/Python311/python.exe "c:/Users/Krishna/OneDrive/Documents/Python/Assignment_2/stu
Student with ID 101 not found.
PS C:\Users\Varish\OneDrive\Documents\Python>]

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.

```
1 import mysql.connector
2 from tabulate import tabulate
3
4 db_conn = mysql.connector.connect(
5     host='localhost',
6     user='root',
7     password='Krishna@128',
8     database='sis'
9 )
10 cursor = db_conn.cursor()
11
12 course_name = "Computer Science 101"
13
14 cursor.execute("""
15     SELECT students.student_id, students.first_name, students.last_name, enrollments.enrollment_date
16     FROM enrollments
17     JOIN students ON enrollments.student_id = students.student_id
18     JOIN courses ON enrollments.course_id = courses.course_id
19     WHERE courses.course_name = %s
20     """, (course_name,))
21 enrollment_records = cursor.fetchall()
22
23 if enrollment_records:
24     report_headers = ["Student ID", "First Name", "Last Name", "Enrollment Date"]
25     enrollment_report = tabulate(enrollment_records, headers=report_headers, tablefmt="pretty")
26
27     print(f"Enrollment Report for {course_name}:\n")
28     print(enrollment_report)
29
30     with open(f"{course_name}_enrollment_report.txt", "w") as file:
31         file.write(f"Enrollment Report for {course_name}:\n\n")
32         file.write(enrollment_report)
33
34 else:
35     print(f"No enrollment records found for {course_name}.")
36
37 cursor.close()
38 db_conn.close()
```