Student Information System – Python Assignment

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.

Code For task 1 and 2

student.py

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

course.py

```python
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
```

teacher.py

```python
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
```

enrollment.py

```python
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
```

payment.py

```python
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
```

sis.py

```python
class Sis:
    def __init__(self):
        self.students = {}
        self.teachers = {}
        self.courses = {}
        self.enrollments ={}
        self.payments = {}
```

Task 3: Implement Methods

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

Student Class:

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

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

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

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

• GetEnrolledCourses(): Retrieves a list of courses in which the student is enrolled.

• GetPaymentHistory(): Retrieves a list of payment records for the student.

Code

student.py

```
from payment import Payment
class Student:  2 usages
    def __init__(self, student_id, first_name, last_name, date_of_birth, email, phone_number):
        self.student_id = student_id
        self.first_name = first_name
        self.last_name = last_name
        self.date_of_birth = date_of_birth
        self.email = email
        self.phone_number = phone_number
        self.enrollments = []
        self.payments = []

    def enroll_in_course(self, course):  3 usages (1 dynamic)
        self.enrollments.append(course)

    def update_student_info(self, first_name, last_name, date_of_birth, email, phone_number):  1 usage
        self.first_name = first_name
        self.last_name = last_name
        self.date_of_birth = date_of_birth
        self.email = email
        self.phone_number = phone_number
```

```python
    def make_payment(self, payment_id, amount, payment_date):  3 usages (1 dynamic)
        payment = Payment(payment_id, self.student_id, amount, payment_date)
        self.payments.append(payment)

    def display_student_info(self):  3 usages
        print(f"ID: {self.student_id}, Name: {self.first_name} {self.last_name}, DOB: {self.date_of_birth}")
        print(f"Email: {self.email}, Phone: {self.phone_number}")

    def get_enrolled_courses(self):  1 usage
        return self.enrollments

    def get_payment_history(self):  2 usages (1 dynamic)
        return self.payments
```

main.py

```python
from student import Student
from datetime import date

def main():  1 usage
    #display info
    student = Student( student_id: 1, first_name: "Tejashree", last_name: "Ganesan", date_of_birth: "2003-08-16", email: "tejashreeganesan@gmail.com", phone_number: "90764378")
    student.display_student_info()

    #Enroll courses
    student.enroll_in_course("Computer Science")
    student.enroll_in_course("DBMS")
    print("\nEnrolled Courses:")
    for course in student.get_enrolled_courses():
        print("-", course)

    student.make_payment( payment_id: 1, amount: 250.00, str(date.today()))
    student.make_payment( payment_id: 2, amount: 350.00, str(date.today()))

    print("\nPayment History:")
    for payment in student.get_payment_history():
        print(payment)

    print("\nBefore Update:")
    student.display_student_info()
    student.update_student_info( first_name: "Nithish", last_name: "Ganesan", date_of_birth: "2000-02-02", email: "nithishganesan@gmail.com", phone_number: "9876543210")
    print("\nAfter Update:")
    student.display_student_info()

if __name__ == "__main__":
    main()
```

Output

```
ID: 1, Name: Tejashree Ganesan, DOB: 2003-08-16
Email: tejashreeganesan@gmail.com, Phone: 90764378

Enrolled Courses:
- Computer Science
- DBMS

Payment History:
PaymentID: 1, StudentID: 1, Amount: $250.0, Date: 2025-06-23
PaymentID: 2, StudentID: 1, Amount: $350.0, Date: 2025-06-23

Before Update:
ID: 1, Name: Tejashree Ganesan, DOB: 2003-08-16
Email: tejashreeganesan@gmail.com, Phone: 90764378

After Update:
ID: 1, Name: Nithish Ganesan, DOB: 2000-02-02
Email: nithishganesan@gmail.com, Phone: 9876543210
```

Course Class:

• AssignTeacher(teacher: Teacher): Assigns a teacher to the course.

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

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

• GetEnrollments(): Retrieves a list of student enrollments for the course.

• GetTeacher(): Retrieves the assigned teacher for the course.

course.py

```
class Course:  2 usages
    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 = []

    def assign_teacher(self, teacher):  1 usage
        self.instructor_name = f"{teacher.first_name} {teacher.last_name}"

    def update_course_info(self, course_code, course_name, instructor):  1 usage
        self.course_code = course_code
        self.course_name = course_name
        self.instructor_name = instructor

    def display_course_info(self):  2 usages
        print(f"Course: {self.course_name} ({self.course_code}), Instructor: {self.instructor_name}")

    def add_enrollment(self, enrollment):  1 usage
        self.enrollments.append(enrollment)

    def get_enrollments(self):  1 usage
        return self.enrollments

    def get_teacher(self):  1 usage
        return self.instructor_name
```

Main.py

```python
from course import Course
from teacher import Teacher
from student import Student
from datetime import date
from enrollment import Enrollment
def main():  1 usage
    # Create a teacher
    teacher = Teacher( teacher_id: "1",  first_name: "Olivia",  last_name: "Black",  email: "olivia@gmail.com",  expertise: "Computer Science")

    # Create a course
    course = Course( course_id: 101,  course_name: "Computer science 101",  course_code: "CS101",  instructor_name: "Olivia Black")
    # Assign teacher to the course
    course.assign_teacher(teacher)
    course.display_course_info()

    # Update course information
    course.update_course_info( course_code: "CS102",  course_name: "Advanced Computer Science",  instructor: "Professor Willword")

    # Retrieve enrollments
    student = Student( student_id: 1,  first_name: "Tejashree",  last_name: "Ganesan",  date_of_birth: "2003-08-16",  email: "tejashreeganesan@gmail.com",  phone_number: "90764378")
    enrollment = Enrollment( enrollment_id: 1, student, course, date.today())
    course.add_enrollment(enrollment)
    student.enrollments.append(enrollment)

    course.display_course_info()
    print("\nEnrollments:")
    for e in course.get_enrollments():
        print(e)

    # Get assigned teacher
    teacher_name = course.get_teacher()
    print("Assigned Teacher:", teacher_name)

if __name__ == "__main__":
    main()
```

Output

```
Course: Computer science 101 (CS101), Instructor: Olivia Black
Course: Advanced Computer Science (CS102), Instructor: Professor Willword

Enrollments:
Tejashree Ganesan enrolled in Advanced Computer Science on 2025-06-23
Assigned Teacher: Professor Willword
```

Enrollment Class:

• GetStudent(): Retrieves the student associated with the enrollment.

• GetCourse(): Retrieves the course associated with the enrollment.

enrollment.py

```python
class Enrollment:  4 usages
    def __init__(self, enrollment_id, student, course, enrollment_date):
        self.enrollment_id = enrollment_id
        self.student = student
        self.course = course
        self.enrollment_date = enrollment_date

    def get_student(self):  1 usage
        return self.student

    def get_course(self):  1 usage
        return self.course

    def __str__(self):
        return f"{self.student.first_name} {self.student.last_name} enrolled in {self.course.course_name} on {self.enrollment_date}"
```

main.py

```python
from student import Student
from course import Course
from teacher import Teacher
from enrollment import Enrollment
from datetime import date
def main():  1 usage
    teacher = Teacher( teacher_id: "1",  first_name: "Olivia",  last_name: "Black",  email: "olivia@gmail.com",  expertise: "Computer Science")
    course = Course( course_id: 101,  course_name: "Computer Science 101",  course_code: "CS101",  instructor_name: "Olivia")
    student = Student( student_id: 1,  first_name: "Tejashree",  last_name: "Ganesan",  date_of_birth: "2003-08-16",  email: "tejashreeganesan@gmail.com",  phone_number: "90764378")
    enrollment = Enrollment( enrollment_id: 1, student, course, date.today())
    enrolled_student = enrollment.get_student()
    enrolled_course = enrollment.get_course()

    print("Retrieved from enrollment:")
    print("Student:", enrolled_student.first_name, enrolled_student.last_name)
    print("Course:", enrolled_course.course_name)

    print("\nEnrollment Summary:")
    print(enrollment)

if __name__ == "__main__":
    main()
```

Output

```
Retrieved from enrollment:
Student: Tejashree Ganesan
Course: Computer Science 101

Enrollment Summary:
Tejashree Ganesan enrolled in Computer Science 101 on 2025-06-23
```

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.

teacher.py

```python
class Teacher:  2 usages
    def __init__(self, teacher_id, first_name, last_name, email, expertise = None):
        self.teacher_id = teacher_id
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.expertise = expertise
        self.assigned_courses = []

    def update_teacher_info(self, first_name, last_name, email, expertise):
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.expertise = expertise

    def display_teacher_info(self):
        print(f"ID: {self.teacher_id}, Name: {self.first_name} {self.last_name}")
        print(f"Email: {self.email}")
        print(f"Expertise: {self.expertise}")

    def get_assigned_courses(self):
        return self.assigned_courses
```

main.py

```python
from course import Course
from teacher import Teacher
def main():  1 usage
    teacher = Teacher( teacher_id: "1",  first_name: "Olivia",  last_name: "Black",  email: "olivia@gmail.com",  expertise: "Computer Science")
    course = Course( course_id: 101,  course_name: "Computer science 101",  course_code: "CS101",  instructor_name: "Olivia Black")
    course.assign_teacher(teacher)
    course.display_course_info()

    # Update course information
    course.update_course_info( course_code: "CS102",  course_name: "Advanced Computer Science",  instructor: "Professor Willword")
    course.display_course_info()

    # Get assigned teacher
    teacher_name = course.get_teacher()
    print("Assigned Teacher:", teacher_name)


if __name__ == "__main__":
    main()
```

Output

```
Course: Computer science 101 (CS101), Instructor: Olivia Black
Course: Advanced Computer Science (CS102), Instructor: Professor Willword
Assigned Teacher: Professor Willword
```

Payment Class:

• GetStudent(): Retrieves the student associated with the payment.

• GetPaymentAmount(): Retrieves the payment amount.

• GetPaymentDate(): Retrieves the payment date.

payment.py

```python
class Payment:  6 usages
    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

    def get_student(self):  1 usage
        return self.student_id

    def get_payment_amount(self):  1 usage
        return self.amount

    def get_payment_date(self):  1 usage
        return self.payment_date

    def __str__(self):
        return f"PaymentID: {self.payment_id}, StudentID: {self.student_id}, Amount: ${self.amount}, Date: {self.payment_date}"
```

main.py

```python
from payment import Payment
from datetime import date
def main():  1 usage
    payment1 = Payment( payment_id: 101,  student_id: 1,  amount: 250.00, date( year: 2024,  month: 6,  day: 1))
    print("\n--- Payment Details ---")
    print(f"Student ID (via get_student): {payment1.get_student()}")
    print(f"Amount (via get_payment_amount): ${payment1.get_payment_amount()}")
    print(f"Date (via get_payment_date): {payment1.get_payment_date()}")

if __name__ == "__main__":
    main()
```

Output

```
--- Payment Details ---
Student ID (via get_student): 1
Amount (via get_payment_amount): $250.0
Date (via get_payment_date): 2024-06-01
```

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.

sis.py

```python
from datetime import datetime
from payment import Payment
from enrollment import Enrollment

class Sis:  2 usages
    def __init__(self):
        self.students = {}
        self.teachers = {}
        self.courses = {}
        self.enrollments = {}
        self.payments = {}

    def enroll_student_in_course(self,enrollment_id, student, course):  1 usage
        enrollment = Enrollment(enrollment_id, student, course,datetime.now().date())
        self.enrollments[enrollment_id]= enrollment
        student.enroll_in_course(enrollment)
        course.add_enrollment(enrollment)
        print(f"{student.first_name} enrolled in {course.course_name} successfully.")

    def assign_teacher_to_course(self, teacher, course):  1 usage
        course.assign_teacher(teacher)
        teacher.assigned_courses.append(course)
        self.teachers[teacher.teacher_id] = teacher
        self.courses[course.course_id] = course
        print(f"{teacher.first_name} {teacher.last_name} assigned to {course.course_name} successfully.")

    def record_payment(self, payment_id, student, amount, payment_date):  1 usage
        payment = Payment(payment_id,student.student_id, amount,payment_date)
        self.payments[payment_id] = payment
        student.make_payment(payment_id, amount, payment_date)
        print(f"Payment of ${amount} by {student.first_name}{student.last_name} recorded successfully.")
```

```python
def generate_enrollment_report(self, course):  1 usage
    print(f"\n -----Enrollment Report for {course.course_name} {course.course_code}-----")
    print(f"Instructor: {course.instructor_name}")
    print(f"Total Enrollment: {len(course.get_enrollments())}")

    if not course.get_enrollments():
        print("No enrollments yet.")
        return
    for enrollment in course.get_enrollments():
        student = enrollment.student
        print(f"Student ID: {student.student_id}, Name: {student.first_name} {student.last_name}, DOB: {student.date_of_birth}, "
              f"Email: {student.email}, Phone Number: {student.phone_number}")

def generate_payment_report(self, student):  1 usage
    print(f"\n -----Payment Report for {student.first_name} {student.last_name} (ID: {student.student_id})-----")
    payments = student.get_payment_history()
    if not payments:
        print("No payments yet.")
        return
    total_paid = 0
    for payment in payments:
        print(f"Payment ID: {payment.payment_id}, Amount: ${payment.amount}, Date: {payment.payment_date}")
        total_paid += payment.amount
    print(f"\n Total Amount Paid: ${total_paid}")
```

```python
def calculate_course_statistics(self, course):  1 usage
    enrollments = course.get_enrollments()
    num_of_enrollments = len(enrollments)
    total_payment = 0
    for enrollment in enrollments:
        student = enrollment.student
        for payment in student.get_payment_history():
            total_payment += payment.amount

    print(f"\n--- Statistics for {course.course_name} ({course.course_code}) ---")
    print(f"Instructor: {course.get_teacher()}")
    print(f"Total Enrollments: {num_of_enrollments}")
    print(f"Total Payments Received: ${total_payment}")
```

main.py

```python
from datetime import datetime
from student import Student
from course import Course
from teacher import Teacher
from sis import Sis
def main():  1 usage
    sis = Sis()
    student = Student( student_id: "1",  first_name: "Tejashree",  last_name: "Ganesan",  date_of_birth: "2003-08-16",  email: "tejashree@gmail.com",  phone_number: "78903552")
    course = Course( course_id: 1,  course_name: "Introduction to programming",  course_code: "CS001",  instructor_name: "Olivia Thompson")
    print("enroll student in course:")
    sis.enroll_student_in_course(enrollment_id=1, student=student, course=course)
    print()
    print("Assign a Teacher to course:")
    teacher = Teacher( teacher_id: 1,  first_name: "Olivia",  last_name: "Thompson",  email: "olivia@gmail.com",  expertise: "Computer Science")
    sis.assign_teacher_to_course(teacher=teacher, course=course)
    print()
    print("Record payment:")
    payment_id = 1
    amount = 500.00
    payment_date = datetime.now().date()
    sis.record_payment(payment_id, student, amount, payment_date)
    print("payment history")
    for payment in student.get_payment_history():
        print(payment)
    print()

    sis.generate_enrollment_report(course)
    sis.generate_payment_report(student)
    sis.calculate_course_statistics(course)

if __name__ == "__main__":
    main()
```

Output

```
enroll student in course:
Tejashree enrolled in Introduction to programming successfully.

Assign a Teacher to course:
Olivia Thompson assigned to Introduction to programming successfully.

Record payment:
Payment of $500.0 by TejashreeGanesan recorded successfully.
payment history
PaymentID: 1, StudentID: 1, Amount: $500.0, Date: 2025-06-24


 -----Enrollment Report for Introduction to programming CS001-----
Instructor: Olivia Thompson
Total Enrollment: 1
Student ID: 1, Name: Tejashree Ganesan, DOB: 2003-08-16, Email: tejashree@gmail.com, Phone Number: 78903552

 -----Payment Report for Tejashree Ganesan (ID: 1)-----
Payment ID: 1, Amount: $500.0, Date: 2025-06-24

 Total Amount Paid: $500.0

--- Statistics for Introduction to programming (CS001) ---
Instructor: Olivia Thompson
Total Enrollments: 1
Total Payments Received: $500.0
```

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.

exception.py

```
class DuplicateEnrollmentException(Exception):  4 usages
    def __init__(self, message = "Student is already enrolled in this course"):
        super().__init__(message)
```

sis.py

```
def enroll_student_in_course(self,enrollment_id, student, course):  2 usages
    for enrollment in self.enrollments.values():
        if enrollment.student.student_id == student.student_id and enrollment.course.course_id == course.course_id:
            raise DuplicateEnrollmentException()

    enrollment = Enrollment(enrollment_id, student, course,datetime.now().date())
    self.enrollments[enrollment_id]= enrollment
    student.enroll_in_course(enrollment)
    course.add_enrollment(enrollment)
    print(f"{student.first_name} enrolled in {course.course_name} successfully.")
```

main.py

```python
from models.sis import Sis
from models.student import Student
from models.course import Course
from exception.exceptions import DuplicateEnrollmentException

sis = Sis()
student = Student( student_id: 1,  first_name: "Tejashree",  last_name: "Ganesan",  date_of_birth: "2003-08-16",  email: "tejashree@gmail.com",
                  phone_number: "897675445")
course = Course( course_id: 1,  course_name: "Introduction to programming",  course_code: "Cs101",  instructor_name: "Olivia Thompson")

sis.students[student.student_id] = student
sis.courses[course.course_id] = course
sis.enroll_student_in_course( enrollment_id: 1, student, course)

try:
    sis.enroll_student_in_course( enrollment_id: 2,student,course)
except DuplicateEnrollmentException as e:
    print(f" Exception Caught: {e}")
```

output

```
Tejashree enrolled in Introduction to programming successfully.
 Exception Caught: Student is already enrolled in this course
```

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).

exception.py

```python
class CourseNotFoundException(Exception):  5 usages
    def __init__(self, message = "This course does not exist in the system"):
        super().__init__(message)
```

sis.py

```python
from datetime import datetime
from exception.exceptions import DuplicateEnrollmentException, CourseNotFoundException
from models.payment import Payment
from models.enrollment import Enrollment
```

```python
def enroll_student_in_course(self,enrollment_id, student, course):  1 usage
    if course.course_id not in self.courses:
        raise CourseNotFoundException()

    for enrollment in self.enrollments.values():
        if enrollment.student.student_id == student.student_id and enrollment.course.course_id == course.course_id:
            raise DuplicateEnrollmentException()

    enrollment = Enrollment(enrollment_id, student, course,datetime.now().date())
    self.enrollments[enrollment_id]= enrollment
    student.enroll_in_course(enrollment)
    course.add_enrollment(enrollment)
    print(f"{student.first_name} enrolled in {course.course_name} successfully.")

def assign_teacher_to_course(self, teacher, course):
    if course.course_id not in self.courses:
        raise CourseNotFoundException()

    course.assign_teacher(teacher)
    teacher.assigned_courses.append(course)
    self.teachers[teacher.teacher_id] = teacher
    self.courses[course.course_id] = course
    print(f"{teacher.first_name} {teacher.last_name} assigned to {course.course_name} successfully.")
```

main.py

```python
from models.sis import Sis
from models.student import Student
from models.course import Course
from exception.exceptions import CourseNotFoundException

sis = Sis()
student = Student( student_id: 1,  first_name: "Tejashree",  last_name: "Ganesan",  date_of_birth: "2003-08-16",  email: "tejashree@gmail.com",
                  phone_number: "897675445")
course = Course( course_id: 899,  course_name: "Quantum Computing",  course_code: "Cs189",  instructor_name: "Robert Willson")

sis.students[student.student_id] = student

try:
    sis.enroll_student_in_course( enrollment_id: 1,student,course)
except CourseNotFoundException as e:
    print(f" Exception Caught: {e}")
```

Output

```
 Exception Caught: This course does not exist in the system
```

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).

exception.py

```python
class StudentNotFoundException(Exception):  5 usages
    def __init__(self, message = "Student not found in the system"):
        super().__init__(message)
```

sis.py

```python
from datetime import datetime
from exception.exceptions import (DuplicateEnrollmentException,
                                  CourseNotFoundException,
                                  StudentNotFoundException,
                                  TeacherNotFoundException)
from models.payment import Payment
from models.enrollment import Enrollment
from models.student import Student
```

```python
def enroll_student_in_course(self,enrollment_id, student, course):  1 usage
    if student.student_id not in self.students:
        raise StudentNotFoundException()

    if course.course_id not in self.courses:
        raise CourseNotFoundException()

    for enrollment in self.enrollments.values():
        if enrollment.student.student_id == student.student_id and enrollment.course.course_id == course.course_id:
            raise DuplicateEnrollmentException()

    enrollment = Enrollment(enrollment_id, student, course,datetime.now().date())
    self.enrollments[enrollment_id]= enrollment
    student.enroll_in_course(enrollment)
    course.add_enrollment(enrollment)
    print(f"{student.first_name} enrolled in {course.course_name} successfully.")
```

main.py

```python
from models.sis import Sis
from models.student import Student
from models.course import Course
from exception.exceptions import CourseNotFoundException, StudentNotFoundException

sis = Sis()
student = Student( student_id: 1,  first_name: "Tejashree",  last_name: "Ganesan",  date_of_birth: "2003-08-16",  email: "tejashree@gmail.com",
                  phone_number: "897675445")
course = Course( course_id: 899,  course_name: "Quantum Computing",  course_code: "Cs189",  instructor_name: "Robert Willson")
sis.courses[course.course_id] = course


try:
    sis.enroll_student_in_course( enrollment_id: 1,student,course)
except StudentNotFoundException as e:
    print(f" Exception Caught: {e}")
```

Output

```
Exception Caught: Student not found in the system
```

TeacherNotFoundException:

exception.py

```
class TeacherNotFoundException(Exception):  4 usages
    def __init__(self, message="Teacher does not exist in the system."):
        super().__init__(message)
```

sis.py

```
from datetime import datetime
from exception.exceptions import (DuplicateEnrollmentException,
                                  CourseNotFoundException,
                                  StudentNotFoundException,
                                  TeacherNotFoundException)
from models.payment import Payment
from models.enrollment import Enrollment
```

```
def assign_teacher_to_course(self, teacher, course):  1 usage
    if course.course_id not in self.courses:
        raise CourseNotFoundException()

    if teacher.teacher_id not in self.teachers:
        raise TeacherNotFoundException()

    course.assign_teacher(teacher)
    teacher.assigned_courses.append(course)
    self.teachers[teacher.teacher_id] = teacher
    self.courses[course.course_id] = course
    print(f"{teacher.first_name} {teacher.last_name} assigned to {course.course_name} successfully.")
```

main.py

```
from models.sis import Sis
from models.student import Student
from models.course import Course
from models.teacher import Teacher
from exception.exceptions import (CourseNotFoundException, StudentNotFoundException,
                                  TeacherNotFoundException)

sis = Sis()
teacher = Teacher( teacher_id: 1,  first_name: "William",  last_name: "Hustin",  email: "william@gmail.com",  expertise: "AI")
course = Course( course_id: 101,  course_name: 'AI Basics',  course_code: 'CS101',  instructor_name: "Professor William")
sis.courses[course.course_id]=course

try:
    sis.assign_teacher_to_course(teacher,course)
except TeacherNotFoundException as e:
    print(f" Exception Caught: {e}")
```

output

```
Exception Caught: Teacher does not exist in the system.
```

PaymentValidationException:

exception.py

```
class PaymentValidationException(Exception):  6 usages
    def __init__(self, message="Invalid payment details provided."):
        super().__init__(message)
```

sis.py

```
def record_payment(self, payment_id, student, amount, payment_date):  2 usages
    if student.student_id not in self.students:
        raise StudentNotFoundException()

    if amount <= 0:
        raise PaymentValidationException("Payment amount must be greater than 0.")
    if not isinstance(payment_date, date):
        raise PaymentValidationException("Payment date must be a valid date.")

    payment = Payment(payment_id,student.student_id, amount,payment_date)
    self.payments[payment_id] = payment
    student.make_payment(payment_id, amount, payment_date)
    print(f"Payment of ${amount} by {student.first_name}{student.last_name} recorded successfully.")
```

main.py

```
from models.sis import Sis
from models.student import Student
from models.course import Course
from models.teacher import Teacher
from exception.exceptions import (CourseNotFoundException, StudentNotFoundException,
                                  TeacherNotFoundException,
                                  PaymentValidationException)
from datetime import datetime,date

sis = Sis()
student = Student( student_id: 1,  first_name: "Tejashree",  last_name: "Ganesan",  date_of_birth: "2003-08-16",  email: "tejashree@gmail.com",  phone_number: "897675445")
sis.students[student.student_id] = student

# Test with invalid amount
try:
    sis.record_payment( payment_id: 1, student, -100, datetime.now().date())  # Invalid amount
except PaymentValidationException as e:
    print(f"Exception Caught: {e}")
except StudentNotFoundException as e:
    print(f"Exception Caught: {e}")

# Test with invalid date
try:
    sis.record_payment( payment_id: 2, student,  amount: 100,  payment_date: "2024-12-12")
except PaymentValidationException as e:
    print(f"Exception Caught: {e}")
```

Output

```
Exception Caught: Payment amount must be greater than 0.
Exception Caught: Payment date must be a valid date.
```

InvalidStudentDataException:

exception.py

```python
class InvalidStudentDataException(Exception):  6 usages
    def __init__(self, message="Invalid student data provided."):
        super().__init__(message)
```

student.py

```python
from .payment import Payment
from exception.exceptions import InvalidStudentDataException
import re
import datetime
class Student:  3 usages
    def __init__(self, student_id, first_name, last_name, date_of_birth, email, phone_number):

        if not re.match( pattern: r"[^@]+@[^@]+\.[^@]+", email):
            raise InvalidStudentDataException("Invalid email format.")

        if isinstance(date_of_birth, str):
            try:
                datetime.datetime.strptime(date_of_birth,  format: "%Y-%m-%d")
            except ValueError:
                raise InvalidStudentDataException("Invalid date of birth format. Use YYYY-MM-DD.")
        elif not isinstance(date_of_birth, datetime.date):
            raise InvalidStudentDataException("Invalid date of birth type. Must be a string or datetime.date.")
```

main.py

```python
from models.sis import Sis
from models.student import Student
from exception.exceptions import InvalidStudentDataException

sis = Sis()

try:
    student = Student( student_id: 1,  first_name: "Tejashree",  last_name: "Ganesan", 2003-16-16,  email: "tej%.com",  phone_number: "897675445")
except InvalidStudentDataException as e:
    print("Exception Caught:", e)
```

Output

```
C:\Users\Admin\AppData\Local\Programs\Python\Python311\python.exe E:\student_information_system\main.py
Exception Caught: Invalid email format.
```

```
Exception Caught: Invalid email format.
```

```
C:\Users\Admin\AppData\Local\Programs\Python\Python311\python.exe E
Exception Caught: Invalid date of birth format. Use YYYY-MM-DD.
```

InvalidCourseDataException:

exception.py

```python
class InvalidCourseDataException(Exception):  5 usages
    def __init__(self, message="Invalid course data provided."):
        super().__init__(message)
```

course.py

```python
from exception.exceptions import InvalidCourseDataException
import re

class Course:  2 usages
    def __init__(self, course_id, course_name, course_code, instructor_name):
        if not re.match( pattern: r"^[A-Z]{2,4}\d{3}$", course_code):
            raise InvalidCourseDataException("Invalid course code format. Example: CS101")

        if not isinstance(instructor_name, str) or not instructor_name.strip():
            raise InvalidCourseDataException("Instructor name cannot be empty.")
```

main.py

```python
from models.course import Course
from exception.exceptions import InvalidCourseDataException

try:
    course = Course( course_id: 1,  course_name: "Intro to AI",  course_code: "AI101",  instructor_name: "")
except InvalidCourseDataException as e:
    print("Exception Caught:", e)
```

Output

```
Exception Caught: Instructor name cannot be empty.
```

Main.py

```python
from models.course import Course
from exception.exceptions import InvalidCourseDataException

try:
    course = Course( course_id: 1,  course_name: "Intro to AI",  course_code: "ci80&",  instructor_name: "Olivia")
except InvalidCourseDataException as e:
    print("Exception Caught:", e)
```

Output

```
Exception Caught: Invalid course code format. Example: CS101
```

InvalidEnrollmentDataException:

exception.py

```python
class InvalidEnrollmentDataException(Exception):  4 usages
    def __init__(self, message="Invalid enrollment data. Student or Course is missing."):
        super().__init__(message)
```

enrollment.py

```python
from exception.exceptions import InvalidEnrollmentDataException

class Enrollment:  4 usages
    def __init__(self, enrollment_id, student, course, enrollment_date):
        if student is None or course is None:
            raise InvalidEnrollmentDataException()
        self.enrollment_id = enrollment_id
        self.student = student
        self.course = course
        self.enrollment_date = enrollment_date
```

main.py

```python
from models.sis import Sis
from models.enrollment import Enrollment
from exception.exceptions import InvalidEnrollmentDataException

from datetime import datetime

try:
    enrollment = Enrollment( enrollment_id: 501, student=None, course=None, enrollment_date=datetime.now().date())
except InvalidEnrollmentDataException as e:
    print("Exception Caught:", e)
```

output

```
Exception Caught: Invalid enrollment data. Student or Course is missing.
```

InvalidTeacherDataException:

exception.py

```python
class InvalidTeacherDataException(Exception):  5 usages
    def __init__(self, message="Invalid teacher data. Name or email is missing."):
        super().__init__(message)
```

teacher.py

```python
from exception.exceptions import InvalidTeacherDataException

class Teacher:  2 usages
    def __init__(self, teacher_id, first_name, last_name, email, expertise = None):
        if not first_name or not last_name or not email:
            raise InvalidTeacherDataException()
        self.teacher_id = teacher_id
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.expertise = expertise
        self.assigned_courses = []

    def update_teacher_info(self, first_name, last_name, email, expertise):  1 usage
        if not first_name or not last_name or not email:
            raise InvalidTeacherDataException("Cannot update: missing first name, last name, or email.")
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.expertise = expertise
```

main.py

```python
from models.teacher import Teacher
from exception.exceptions import InvalidTeacherDataException

try:
    teacher = Teacher( teacher_id: 101,  first_name: "Tejashree",  last_name: "Ganesan",  email: "",  expertise: "Computer Science")
except InvalidTeacherDataException as e:
    print("Exception Caught:", e)
```

output

```
Exception Caught: Invalid teacher data. Name or email is missing.
```

main.py

```python
from models.teacher import Teacher
from exception.exceptions import InvalidTeacherDataException


teacher = Teacher( teacher_id: 101,  first_name: "Olivia",  last_name: "Tomson",  email: "olivia@gmail.com",  expertise: "CS")
try:
    teacher.update_teacher_info( first_name: "Oliva",  last_name: "",  email: "",  expertise: "Math")
except InvalidTeacherDataException as e:
    print("Exception Caught:", e)
```

Output

```
Exception Caught: Cannot update: missing first name, last name, or email.
```

InsufficientFundsException:

exception.py

```python
class InsufficientFundsException(Exception):  4 usages
    def __init__(self, message="Student does not have sufficient funds to enroll."):
        super().__init__(message)
```

sis.py

```python
def enroll_student_in_course(self,enrollment_id, student, course,  required_amount=100):  1 usage
    if student.balance < required_amount:
        raise InsufficientFundsException()
    student.balance -= required_amount
    if student.student_id not in self.students:
        raise StudentNotFoundException()
```

student.py

```python
class Student:  3 usages
    def __init__(self, student_id, first_name, last_name, date_of_birth, email, phone_number, balance=0):
        if not re.match( pattern: r"[^@]+@[^@]+\.[^@]+", email):
            raise InvalidStudentDataException("Invalid email format.")
        if isinstance(date_of_birth, str):
            try:
                datetime.datetime.strptime(date_of_birth,  format: "%Y-%m-%d")
            except ValueError:
                raise InvalidStudentDataException("Invalid date of birth format. Use YYYY-MM-DD.")
        elif not isinstance(date_of_birth, datetime.date):
            raise InvalidStudentDataException("Invalid date of birth type. Must be a string or datetime.date.")

        self.student_id = student_id
        self.first_name = first_name
        self.last_name = last_name
        self.date_of_birth = date_of_birth
        self.email = email
        self.phone_number = phone_number
        self.enrollments = []
        self.payments = []
        self.balance = balance
```

main.py

```python
from models.sis import Sis
from models.student import Student
from models.course import Course
from exception.exceptions import InsufficientFundsException

sis = Sis()

student = Student( student_id: 1,  first_name: "Teja",  last_name: "Ganesan",  date_of_birth: "2003-08-16",  email: "teja@gmail.com",  phone_number: "9876543210", balance=50)
course = Course( course_id: 1,  course_name: "AI & ML",  course_code: "CS500",  instructor_name: "Dr. Rao")

sis.students[student.student_id] = student
sis.courses[course.course_id] = course

try:
    sis.enroll_student_in_course( enrollment_id: 101, student, course, required_amount=100)
except InsufficientFundsException as e:
    print("Exception Caught:", e)
```

Output

```
Exception Caught: Student does not have sufficient funds to enroll.
```

Task 5: Collections

Implement Collections: Implement relationships between classes using appropriate data structures (e.g., lists or dictionaries) to maintain associations between students, courses, enrollments, teachers, and payments.

These relationships are essential for the Student Information System (SIS) to track and manage student enrollments, teacher assignments, and payments accurately.

Define Class-Level Data Structures You will need class-level data structures within each class to maintain relationships. Here's how to define them for each class:

Student Class:

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

Example: List Enrollments { get; set; }

student.py

```python
import datetime
class Student:  3 usages
    def __init__(self, student_id, first_name, last_name, date_of_birth, email, phone_number, balance=0):
        if not re.match( pattern: r"[^@]+@[^@]+\.[^@]+", email):
            raise InvalidStudentDataException("Invalid email format.")
        if isinstance(date_of_birth, str):
            try:
                datetime.datetime.strptime(date_of_birth,  format: "%Y-%m-%d")
            except ValueError:
                raise InvalidStudentDataException("Invalid date of birth format. Use YYYY-MM-DD.")
        elif not isinstance(date_of_birth, datetime.date):
            raise InvalidStudentDataException("Invalid date of birth type. Must be a string or datetime.date.")

        self.student_id = student_id
        self.first_name = first_name
        self.last_name = last_name
        self.date_of_birth = date_of_birth
        self.email = email
        self.phone_number = phone_number
        self.enrollments = [] #List of Enrollment objects
        self.payments = [] #List of Payments objects
        self.balance = balance
```

Course Class:

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

Example: List Enrollments { get; set; }

course.py

```python
class Course:  2 usages
    def __init__(self, course_id, course_name, course_code, instructor_name):
        if not re.match( pattern: r"^[A-Z]{2,4}\d{3}$", course_code):
            raise InvalidCourseDataException("Invalid course code format. Example: CS101")

        if not isinstance(instructor_name, str) or not instructor_name.strip():
            raise InvalidCourseDataException("Instructor name cannot be empty.")

        self.course_id = course_id
        self.course_name = course_name
        self.course_code = course_code
        self.instructor_name = instructor_name
        self.enrollments = [] # List of Enrollment objects
```

Enrollment Class:

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

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

enrollment.py

```python
class Enrollment:  2 usages
    def __init__(self, enrollment_id, student, course, enrollment_date):
        if student is None or course is None:
            raise InvalidEnrollmentDataException()
        self.enrollment_id = enrollment_id
        self.student = student # holds student object
        self.course = course #holds course object
        self.enrollment_date = enrollment_date
```

Teacher Class:

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

Example: List AssignedCourses { get; set; }

teacher.py

```python
class Teacher:
    def __init__(self, teacher_id, first_name, last_name, email, expertise = None):
        if not first_name or not last_name or not email:
            raise InvalidTeacherDataException()
        self.teacher_id = teacher_id
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.expertise = expertise
        self.assigned_courses = [] #list of course objects
```

Payment Class:

Include a property to hold a reference to the Student object.

Example: Student Student { get; set; }

payment.py

```python
class Payment:  4 usages
    def __init__(self, payment_id, student_id, amount, payment_date):
        self.payment_id = payment_id
        self.student_id = student_id #student object
        self.amount = amount
        self.payment_date = payment_date
```

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.

sis.py

```python
def add_enrollment(self, student, course, enrollment_date):  3 usages (2 dynamic)
    if student.student_id not in self.students:
        raise StudentNotFoundException()

    if course.course_id not in self.courses:
        raise CourseNotFoundException()

    enrollment_id = len(self.enrollments) + 1

    for enrollment in self.enrollments.values():
        if enrollment.student.student_id == student.student_id and enrollment.course.course_id == course.course_id:
            raise DuplicateEnrollmentException()

    enrollment = Enrollment(enrollment_id, student, course, enrollment_date)
    self.enrollments[enrollment_id] = enrollment

    student.enroll_in_course(enrollment)
    course.add_enrollment(enrollment)

    print(f"Enrollment added: {student.first_name} enrolled in {course.course_name} on {enrollment_date}.")
```

student.py

```python
def enroll_in_course(self, course):  2 usages (2 dynamic)
    self.enrollments.append(course)
```

course.py

```python
def add_enrollment(self, enrollment):  2 usages (2 dynamic)
    if enrollment in self.enrollments:
        print("Enrollment already exists in this course.")
    else:
        self.enrollments.append(enrollment)
```

main.py

```python
from models.sis import Sis
from models.student import Student
from models.course import Course
from exception.exceptions import CourseNotFoundException, StudentNotFoundException, DuplicateEnrollmentException
from datetime import date

sis = Sis()
student = Student( student_id: 1,  first_name: "Tejashree",  last_name: "Ganesan",  date_of_birth: "2003-08-16",  email: "tejashree@gmail.com",
                  phone_number: "897675445")
course = Course( course_id: 101,  course_name: "Python Programming",  course_code: "CS101",  instructor_name: "Dr. Smith")

sis.students[student.student_id] = student
sis.courses[course.course_id] = course

try:
    sis.add_enrollment(student, course, date.today())
except (StudentNotFoundException, CourseNotFoundException, DuplicateEnrollmentException) as e:
    print(f"Exception Caught: {e}")
```

Output

```
Enrollment added: Tejashree enrolled in Python Programming on 2025-06-25.
```

• 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.

sis.py

```
def assign_course_to_teacher(self, course, teacher):  1 usage
    if teacher.teacher_id not in self.teachers:
        raise TeacherNotFoundException("Teacher not found.")

    if course.course_id not in self.courses:
        raise CourseNotFoundException("Course not found.")

    teacher.assigned_courses.append(course)
    course.assign_teacher(teacher)
    print(
        f"Course '{course.course_name}' assigned to teacher {teacher.first_name} {teacher.last_name} successfully.")
```

course.py

```
def assign_teacher(self, teacher):  2 usages (2 dynamic)
    self.instructor_name = f"{teacher.first_name} {teacher.last_name}"
```

teacher.py

```
def get_assigned_courses(self):
    return self.assigned_courses
```

main.py

```
from models.sis import Sis
from models.teacher import Teacher
from models.course import Course
from exception.exceptions import TeacherNotFoundException, CourseNotFoundException

sis = Sis()
teacher = Teacher( teacher_id: 1,  first_name: "Elena",  last_name: "Will",  email: "elena@gmail.com",  expertise: "Data Science")
course = Course( course_id: 101,  course_name: "Machine Learning",  course_code: "DS101",  instructor_name: "Placeholder")

sis.teachers[teacher.teacher_id] = teacher
sis.courses[course.course_id] = course

try:
    sis.assign_course_to_teacher(course, teacher)
except (TeacherNotFoundException, CourseNotFoundException) as e:
    print("Exception Caught:", e)
```

Output

```
Course 'Machine Learning' assigned to teacher Elena Will successfully.
```

• 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.

student.py

```
def make_payment(self, payment_id, amount, payment_date):  2 usages (2 dynamic)
    payment = Payment(payment_id, self.student_id, amount, payment_date)
    self.payments.append(payment)
```

sis.py

```
def add_payment(self, payment_id, student, amount, payment_date):
    if student.student_id not in self.students:
        raise StudentNotFoundException("Student not found in the system.")
    if amount <= 0:
        raise PaymentValidationException("Payment amount must be greater than zero.")
    if not isinstance(payment_date, date):
        raise PaymentValidationException("Invalid payment date format. Must be a date object.")

    payment = Payment(payment_id, student.student_id, amount, payment_date)
    self.payments[payment_id] = payment
    student.make_payment(payment_id, amount, payment_date)

    print(f"Payment of ${amount} added for student {student.first_name} {student.last_name} on {payment_date}.")
```

main.py

```
from models.sis import Sis
from models.student import Student
from datetime import date
from exception.exceptions import StudentNotFoundException, PaymentValidationException

sis = Sis()
student = Student( student_id: 1,  first_name: "Tejashree",  last_name: "Ganesan",  date_of_birth: "2003-08-16",  email: "tejashree@gmail.com",
                  phone_number: "897675445")
sis.students[student.student_id] = student

try:
    sis.add_payment( payment_id: 101, student,  amount: 1500, date.today())
except (StudentNotFoundException, PaymentValidationException) as e:
    print("Exception Caught:", e)
```

Output

```
Payment of $1500 added for student Tejashree Ganesan on 2025-06-25.
```

• GetEnrollmentsForStudent(student):

In the SIS class, create a method to retrieve all enrollments for a specific student.

sis.py

```python
def get_enrollments_for_student(self, student):  1 usage
    if student.student_id not in self.students:
        raise StudentNotFoundException("Student not found in the system.")

    enrollments = []
    for enrollment in self.enrollments.values():
        if enrollment.student.student_id == student.student_id:
            enrollments.append(enrollment)

    return enrollments
```

main.py

```python
from models.sis import Sis
from models.student import Student
from models.course import Course

sis = Sis()

student = Student( student_id: 1,  first_name: "Tejashree",  last_name: "Ganesan",  date_of_birth: "2003-08-16",  email: "tejashree@gmail.com",
                  phone_number: "897675445", balance=300)
course1 = Course( course_id: 101,  course_name: "Python Basics",  course_code: "CS101",  instructor_name: "Mr. Kumar")
course2 = Course( course_id: 102,  course_name: "Data Science",  course_code: "DS102",  instructor_name: "Ms. Anu")

sis.students[student.student_id] = student
sis.courses[course1.course_id] = course1
sis.courses[course2.course_id] = course2

sis.enroll_student_in_course( enrollment_id: 201, student, course1)
sis.enroll_student_in_course( enrollment_id: 202, student, course2)

print(f"\nEnrollments for {student.first_name} {student.last_name}:")
enrollments = sis.get_enrollments_for_student(student)
for e in enrollments:
    print(f"Enrollment ID: {e.enrollment_id}, Course: {e.course.course_name}, Date: {e.enrollment_date}")
```

Output

```
Tejashree enrolled in Python Basics successfully.
Tejashree enrolled in Data Science successfully.

Enrollments for Tejashree Ganesan:
Enrollment ID: 201, Course: Python Basics, Date: 2025-06-25
Enrollment ID: 202, Course: Data Science, Date: 2025-06-25
```

- **GetCoursesForTeacher(teacher):**

In the SIS class, create a method to retrieve all courses assigned to a specific teacher.

teacher.py

```python
def get_assigned_courses(self):  1 usage (1 dynamic)
    return self.assigned_courses
```

sis.py

```python
def get_courses_for_teacher(self, teacher):  1 usage
    if teacher.teacher_id not in self.teachers:
        raise TeacherNotFoundException(f"Teacher with ID {teacher.teacher_id} not found.")

    courses = teacher.get_assigned_courses()
    print(f"\nCourses assigned to {teacher.first_name} {teacher.last_name}:")
    for course in courses:
        print(f"- {course.course_name} ({course.course_code})")
```

main.py

```python
from models.sis import Sis
from models.teacher import Teacher
from exception.exceptions import TeacherNotFoundException


sis = Sis()
teacher = Teacher( teacher_id: 1,  first_name: "Olivia",  last_name: "Thompson",  email: "olivia@gmail.com",  expertise: "Computer Science")
sis.teachers[teacher.teacher_id] = teacher
course = Course( course_id: 101,  course_name: "Advanced Python",  course_code: "CS301",  instructor_name: "Olivia Thompson")
sis.courses[course.course_id] = course

sis.assign_teacher_to_course(teacher, course)

try:
    sis.get_courses_for_teacher(teacher)
except TeacherNotFoundException as e:
    print("Exception Caught:", e)
```

Output

```
Olivia Thompson assigned to Advanced Python successfully.

Courses assigned to Olivia Thompson:
- Advanced Python (CS301)
```

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.

Util/DBConnection.py

```python
import mysql.connector

class DBConnection:

    @staticmethod
    def get_connection():
        return mysql.connector.connect(
            host="localhost",
            user="root",
            password="Tejashree85!",
            database="sis"
        )
```

DBInitializer.py

```python
from mysql.connector import Error, connection
from util.DBConnection import DBConnection

def initialize_database():  1 usage
    try:
        conn = DBConnection.get_connection()
        cursor = conn.cursor()

        cursor.execute("CREATE DATABASE IF NOT EXISTS sis")
        print("Database 'sis' checked/created.")
        connection.database = "sis"
        cursor.execute("""
        CREATE TABLE IF NOT EXISTS students (
            student_id INT PRIMARY KEY,
            first_name VARCHAR(100),
            last_name VARCHAR(100),
            date_of_birth DATE,
            email VARCHAR(255),
            phone_number VARCHAR(20),
            balance DECIMAL(10, 2)
        )
        """)
```

```
cursor.execute("""
CREATE TABLE IF NOT EXISTS courses (
    course_id INT PRIMARY KEY,
    course_name VARCHAR(255),
    course_code VARCHAR(50),
    instructor_name VARCHAR(255)
)
""")
cursor.execute("""
CREATE TABLE IF NOT EXISTS teachers (
    teacher_id INT PRIMARY KEY,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    email VARCHAR(255),
    expertise VARCHAR(255)
)
""")
cursor.execute("""
CREATE TABLE IF NOT EXISTS enrollments (
    enrollment_id INT PRIMARY KEY,
    student_id INT,
    course_id INT,
    enrollment_date DATE,
    FOREIGN KEY (student_id) REFERENCES students(student_id),
    FOREIGN KEY (course_id) REFERENCES courses(course_id)
)
""")
```

```
            cursor.execute("""
            CREATE TABLE IF NOT EXISTS payments (
                payment_id INT PRIMARY KEY,
                student_id INT,
                amount DECIMAL(10, 2),
                payment_date DATE,
                FOREIGN KEY (student_id) REFERENCES students(student_id)
            )
            """)

            connection.commit()
            print("All tables created successfully.")

        except Error as e:
            print("Error while connecting to MySQL", e)

        finally:
            if connection.is_connected():
                cursor.close()
                connection.close()
                print("MySQL connection closed.")

if __name__ == "__main__":
    initialize_database()
```

Output

```
Database 'sis' checked/created.
All tables created successfully.
MySQL connection closed.
```

Inserting values to student, courses, and Teacher table

```python
from util.DBConnection import DBConnection

def insert_sample_students():  1 usage
    conn = DBConnection.get_connection()
    cursor = conn.cursor()
    try:
        cursor.execute("""
            INSERT INTO students (student_id, first_name, last_name, date_of_birth, email, phone_number, balance)
            VALUES (1, 'Tejashree', 'Ganesan', '2003-08-16', 'tejashree@gmail.com', '8976754455', 0)
        """)
        conn.commit()
        print("Student inserted successfully.")
    except Exception as e:
        print("Error inserting student:", e)
    finally:
        cursor.close()
        conn.close()
```

```python
def insert_sample_teachers():  1 usage
    conn = DBConnection.get_connection()
    cursor = conn.cursor()
    try:
        cursor.execute("""
            INSERT INTO teachers (teacher_id, first_name, last_name, email, expertise)
            VALUES (1, 'Olivia', 'Thompson', 'olivia@gmail.com', 'Computer Science')
        """)
        conn.commit()
        print("Teacher inserted successfully.")
    except Exception as e:
        print("Error inserting teacher:", e)
    finally:
        cursor.close()
        conn.close()
```

```python
def insert_sample_courses():  1 usage
    conn = DBConnection.get_connection()
    cursor = conn.cursor()
    try:
        cursor.execute("""
            INSERT INTO courses (course_id, course_name, course_code, instructor_name)
            VALUES (1, 'Computer Science', 'CS809', 'Olivia Thompson')
        """)
        conn.commit()
        print("Course inserted successfully.")
    except Exception as e:
        print("Error inserting course:", e)
    finally:
        cursor.close()
        conn.close()

if __name__ == "__main__":
    insert_sample_students()
    insert_sample_teachers()
    insert_sample_courses()
```

Output

```
Student inserted successfully.
Teacher inserted successfully.
Course inserted successfully.

Process finished with exit code 0
```

Inserting values to Enrollments and Payments table

```python
from datetime import date
from util.DBConnection import DBConnection

def insert_sample_enrollment():  1 usage
    conn = DBConnection.get_connection()
    cursor = conn.cursor()
    try:
        cursor.execute("""
            INSERT INTO enrollments (enrollment_id, student_id, course_id, enrollment_date)
            VALUES (1, 1, 1, %s)
        """, (date.today(),))
        conn.commit()
        print("Enrollment inserted.")
    except Exception as e:
        print("Error inserting enrollment:", e)
    finally:
        cursor.close()
        conn.close()

def insert_sample_payment():  1 usage
    conn = DBConnection.get_connection()
    cursor = conn.cursor()
    try:
        cursor.execute("""
            INSERT INTO payments (payment_id, student_id, amount, payment_date)
            VALUES (1, 1, 500, %s)
        """, (date.today(),))
        conn.commit()
        print("Payment inserted.")
    except Exception as e:
        print("Error inserting payment:", e)
    finally:
        cursor.close()
        conn.close()

if __name__ == "__main__":
    insert_sample_enrollment()
    insert_sample_payment()
```

Output

```
Enrollment inserted.
Payment inserted.
```

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.

Retriving data from student table

```python
from util.DBConnection import DBConnection

def get_all_students():  1 usage
    conn = DBConnection.get_connection()
    cursor = conn.cursor(dictionary=True)
    try:
        cursor.execute("SELECT * FROM students")
        students = cursor.fetchall()
        if not students:
            print("No students found.")
        else:
            for student in students:
                print(student)
    except Exception as e:
        print("Error fetching students:", e)
    finally:
        cursor.close()
        conn.close()

if __name__ == "__main__":
    print("Students:")
    get_all_students()
```

Output

```
Students:
{'student_id': 1, 'first_name': 'Tejashree', 'last_name': 'Ganesan', 'date_of_birth': datetime.date(2003, 8, 16), 'email': 'tejashree@gmail.com', 'phone_number': '8976754455',
```

Retrieving data from course table

```python
from util.DBConnection import DBConnection

def get_all_courses():  1 usage
    conn = DBConnection.get_connection()
    cursor = conn.cursor(dictionary=True)
    try:
        cursor.execute("SELECT * FROM courses")
        courses = cursor.fetchall()
        if not courses:
            print("No courses found.")
        else:
            for course in courses:
                print(course)
    except Exception as e:
        print("Error fetching courses:", e)
    finally:
        cursor.close()
        conn.close()

if __name__ == "__main__":
    print("Courses:")
    get_all_courses()
```

Output

```
Courses:
{'course_id': 1, 'course_name': 'Computer Science', 'course_code': 'CS809', 'instructor_name': 'Olivia Thompson'}
```

Retrieving data from Teacher table

```python
from util.DBConnection import DBConnection
def get_all_teachers():  1 usage
    conn = DBConnection.get_connection()
    cursor = conn.cursor(dictionary=True)
    try:
        cursor.execute("SELECT * FROM teachers")
        teachers = cursor.fetchall()
        if not teachers:
            print("No teachers found.")
        else:
            for teacher in teachers:
                print(teacher)
    except Exception as e:
        print("Error fetching teachers:", e)
    finally:
        cursor.close()
        conn.close()

if __name__ == "__main__":
    print("Teachers:")
    get_all_teachers()
```

Output

```
Teachers:
{'teacher_id': 1, 'first_name': 'Olivia', 'last_name': 'Thompson', 'email': 'olivia@gmail.com', 'expertise': 'Computer Science'}
```

Retrieving data from Enrollments table

```python
from util.DBConnection import DBConnection

def get_all_enrollments():  1 usage
    conn = DBConnection.get_connection()
    cursor = conn.cursor(dictionary=True)
    try:
        cursor.execute("""
            SELECT e.enrollment_id, s.first_name AS student, c.course_name AS course, e.enrollment_date
            FROM enrollments e
            JOIN students s ON e.student_id = s.student_id
            JOIN courses c ON e.course_id = c.course_id
        """)
        enrollments = cursor.fetchall()
        if not enrollments:
            print("No enrollments found.")
        else:
            for enrollment in enrollments:
                print(enrollment)
    except Exception as e:
        print("Error fetching enrollments:", e)
    finally:
        cursor.close()
        conn.close()

if __name__ == "__main__":
    print("Enrollments:")
    get_all_enrollments()
```

Output

```
{'enrollment_id': 1, 'student': 'Tejashree', 'course': 'Computer Science', 'enrollment_date': datetime.date(2025, 6, 25)}

Process finished with exit code 0
```

Retrieving data from Payments table

```python
from util.DBConnection import DBConnection

def get_all_payments():  1 usage
    conn = DBConnection.get_connection()
    cursor = conn.cursor(dictionary=True)
    try:
        cursor.execute("""
            SELECT p.payment_id, s.first_name AS student, p.amount, p.payment_date
            FROM payments p
            JOIN students s ON p.student_id = s.student_id
        """)
        payments = cursor.fetchall()
        if not payments:
            print("No payments found.")
        else:
            for payment in payments:
                print(payment)
    except Exception as e:
        print("Error fetching payments:", e)
    finally:
        cursor.close()
        conn.close()

if __name__ == "__main__":
    print("Payments:")
    get_all_payments()
```

Output

```
Payments:
{'payment_id': 1, 'student': 'Tejashree', 'amount': Decimal('500.00'), 'payment_date': datetime.date(2025, 6, 25)}
```

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.

Inserting New Enrollment

```python
from util.DBConnection import DBConnection
from datetime import date
def insert_enrollment(enrollment_id, student_id, course_id, enrollment_date):  1 usage
    conn = DBConnection.get_connection()
    cursor = conn.cursor()
    try:
        # Validation
        if not all([student_id, course_id]):
            raise ValueError("Student ID and Course ID must be provided.")

        query = """
            INSERT INTO enrollments (enrollment_id, student_id, course_id, enrollment_date)
            VALUES (%s, %s, %s, %s)
        """
        cursor.execute(query, (enrollment_id, student_id, course_id, enrollment_date))
        conn.commit()
        print("Enrollment inserted successfully.")
    except Exception as e:
        print("Failed to insert enrollment:", e)
    finally:
        cursor.close()
        conn.close()

if __name__ == "__main__":
    insert_enrollment(
        enrollment_id=2,
        student_id=1,
        course_id=2,
        enrollment_date=date( year: 2025,  month: 6,  day: 16)
    )
```

Output

```
Enrollment inserted successfully.
```

	enrollment_id	student_id	course_id	enrollment_date
▶	1	1	1	2025-06-25
	2	1	2	2025-06-16
	NULL	NULL	NULL	NULL

Inserting New Payment

```python
def insert_payment(payment_id, student_id, amount, payment_date):  1 usage
    conn = DBConnection.get_connection()
    cursor = conn.cursor()
    try:
        # Validation
        if amount <= 0:
            raise ValueError("Payment amount must be greater than 0.")
        if not isinstance(payment_date, datetime):
            raise ValueError("Invalid payment date format. It must be a datetime object.")

        query = """
            INSERT INTO payments (payment_id, student_id, amount, payment_date)
            VALUES (%s, %s, %s, %s)
        """
        cursor.execute(query, (payment_id, student_id, amount, payment_date))
        conn.commit()
        print("Payment inserted successfully.")
    except Exception as e:
        print("Failed to insert payment:", e)
    finally:
        cursor.close()
        conn.close()

if __name__ == "__main__":
    insert_payment(
        payment_id=2,
        student_id=1,
        amount=200,
        payment_date=datetime.now()
    )
```

Output

```
Payment inserted successfully.
```

payment_id	student_id	amount	payment_date
1	1	500.00	2025-06-25
2	1	200.00	2025-06-25
NULL	NULL	NULL	NULL

Updating existing Student Information

```python
from util.DBConnection import DBConnection
from datetime import date, datetime

def update_student_info(student_id, first_name, last_name, dob, email, phone):  1 usage
    conn = DBConnection.get_connection()
    cursor = conn.cursor()
    try:
        # Validations
        if not email or "@" not in email:
            raise ValueError("Invalid email format.")
        if not phone.isdigit() or len(phone) < 10:
            raise ValueError("Invalid phone number.")

        query = """
            UPDATE students
            SET first_name = %s, last_name = %s, date_of_birth = %s, email = %s, phone_number = %s
            WHERE student_id = %s
        """
        cursor.execute(query, (first_name, last_name, dob, email, phone, student_id))
        conn.commit()

        if cursor.rowcount == 0:
            print("No student found with that ID.")
        else:
            print("Student info updated successfully.")
    except Exception as e:
        print("Failed to update student:", e)
    finally:
        cursor.close()
        conn.close()

if __name__ == "__main__":
    update_student_info( student_id=1,first_name="Tejashree",last_name="Ganesan",dob="2003-08-16", email="tejashree_updated@gmail.com",phone="9876543210")
```

Output

```
Student info updated successfully.
```

student_id	first_name	last_name	date_of_birth	email	phone_number	balance
1	Tejashree	Ganesan	2003-08-16	tejashree_updated@gmail.com	9876543210	0.00
NULL	NULL	NULL	NULL	NULL	NULL	NULL

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.

Example 1: Enrolling Students to a course that doesn't exist (Rollback mechanism)

```python
from util.DBConnection import DBConnection
from datetime import date, datetime

def enroll_student_with_transaction(student_id, course_id, enrollment_date):  1 usage
    enrollment_id = 4
    try:
        connection = DBConnection.get_connection()
        cursor = connection.cursor()
        connection.start_transaction()
        cursor.execute("SELECT * FROM students WHERE student_id = %s", (student_id,))
        if not cursor.fetchone():
            raise Exception("Student does not exist.")
        cursor.execute("SELECT * FROM courses WHERE course_id = %s", (course_id,))
        if not cursor.fetchone():
            raise Exception("Course does not exist.")
        cursor.execute(
            "INSERT INTO enrollments (enrollment_id, student_id, course_id, enrollment_date) VALUES (%s, %s, %s, %s)",
            (enrollment_id, student_id, course_id, enrollment_date)
        )
        connection.commit()
        print("Enrollment successful and transaction committed.")
    except Exception as e:
        connection.rollback()
        print("Enrollment failed. Rolled back transaction.")
        print("Reason:", e)
    finally:
        cursor.close()
        connection.close()

if __name__ == "__main__":
    enroll_student_with_transaction(student_id=1,course_id=5,enrollment_date=date.today())
```

Output

```
Enrollment failed. Rolled back transaction.
Reason: Course does not exist.
```

Example 2: Enrolling Students in a course that exists

```python
from datetime import date, datetime

def enroll_student_with_transaction(student_id, course_id, enrollment_date):  1 usage
    enrollment_id = 4
    try:
        connection = DBConnection.get_connection()
        cursor = connection.cursor()
        connection.start_transaction()
        cursor.execute("SELECT * FROM students WHERE student_id = %s", (student_id,))
        if not cursor.fetchone():
            raise Exception("Student does not exist.")
        cursor.execute("SELECT * FROM courses WHERE course_id = %s", (course_id,))
        if not cursor.fetchone():
            raise Exception("Course does not exist.")
        cursor.execute(
            "INSERT INTO enrollments (enrollment_id, student_id, course_id, enrollment_date) VALUES (%s, %s, %s, %s)",
            (enrollment_id, student_id, course_id, enrollment_date)
        )
        connection.commit()
        print("Enrollment successful and transaction committed.")
    except Exception as e:
        connection.rollback()
        print("Enrollment failed. Rolled back transaction.")
        print("Reason:", e)
    finally:
        cursor.close()
        connection.close()

if __name__ == "__main__":
    enroll_student_with_transaction(student_id=1,course_id=2,enrollment_date=date.today())
```

Output

```
Enrollment successful and transaction committed.
```

enrollment_id	student_id	course_id	enrollment_date
1	1	1	2025-06-25
2	1	2	2025-06-16
4	1	2	2025-06-25
5	1	2	2025-06-25

Example 3: Assigning teacher to a course that doesn't exists (Rollback Mechanism)

```python
from util.DBConnection import DBConnection
from datetime import date, datetime

def assign_teacher_with_transaction(course_id, teacher_name):  1 usage
    try:
        connection = DBConnection.get_connection()
        cursor = connection.cursor()
        connection.start_transaction()
        cursor.execute("SELECT * FROM courses WHERE course_id = %s", (course_id,))
        if not cursor.fetchone():
            raise Exception("Course does not exist.")
        cursor.execute(
            "UPDATE courses SET instructor_name = %s WHERE course_id = %s",
            (teacher_name, course_id)
        )
        connection.commit()
        print("Teacher assigned successfully.")
    except Exception as e:
        connection.rollback()
        print("Failed to assign teacher. Rolled back.")
        print("Reason:", e)
    finally:
        cursor.close()
        connection.close()

if __name__ == "__main__":
    assign_teacher_with_transaction(course_id=3,teacher_name="William James")
```

Output

```
Failed to assign teacher. Rolled back.
Reason: Course does not exist.
```

Example 4: Assigning a teacher to a course that does exist

```python
from util.DBConnection import DBConnection
from datetime import date, datetime

def assign_teacher_with_transaction(course_id, teacher_name):  1 usage
    try:
        connection = DBConnection.get_connection()
        cursor = connection.cursor()
        connection.start_transaction()
        cursor.execute("SELECT * FROM courses WHERE course_id = %s", (course_id,))
        if not cursor.fetchone():
            raise Exception("Course does not exist.")
        cursor.execute(
            "UPDATE courses SET instructor_name = %s WHERE course_id = %s",
            (teacher_name, course_id)
        )
        connection.commit()
        print("Teacher assigned successfully.")
    except Exception as e:
        connection.rollback()
        print("Failed to assign teacher. Rolled back.")
        print("Reason:", e)
    finally:
        cursor.close()
        connection.close()

if __name__ == "__main__":
    assign_teacher_with_transaction(course_id=1,teacher_name="William James")
```

Output

```
Teacher assigned successfully.
```

	course_id	course_name	course_code	instructor_name
▶	1	Computer Science	CS809	William James
	2	AI	CS102	Olivia Thompson
*	NULL	NULL	NULL	NULL

Example 5: Recording a Payment with student Id that doesn't exists (Rollback Mechanism)

```python
from util.DBConnection import DBConnection
from datetime import date, datetime
def record_payment_with_transaction(payment_id, student_id, amount, payment_date):  1 usage
    try:
        connection = DBConnection.get_connection()
        cursor = connection.cursor()
        connection.start_transaction()
        cursor.execute("SELECT * FROM students WHERE student_id = %s", (student_id,))
        if not cursor.fetchone():
            raise Exception("Student does not exist.")
        if amount <= 6:
            raise Exception("Invalid payment amount.")
        cursor.execute(
            "INSERT INTO payments (payment_id, student_id, amount, payment_date) VALUES (%s, %s, %s, %s)",
            (payment_id, student_id, amount, payment_date)
        )
        connection.commit()
        print("Payment recorded successfully.")
    except Exception as e:
        connection.rollback()
        print("Failed to record payment. Rolled back.")
        print("Reason:", e)
    finally:
        cursor.close()
        connection.close()

if __name__ == "__main__":
    record_payment_with_transaction(payment_id=18,student_id=9,amount=500,payment_date=date.today())
```

Output

```
Failed to record payment. Rolled back.
Reason: Student does not exist.
```

Example 6: Recording a Payment with a student ID that does exist

```python
from util.DBConnection import DBConnection
from datetime import date, datetime
def record_payment_with_transaction(payment_id, student_id, amount, payment_date):  1 usage
    try:
        connection = DBConnection.get_connection()
        cursor = connection.cursor()
        connection.start_transaction()
        cursor.execute("SELECT * FROM students WHERE student_id = %s", (student_id,))
        if not cursor.fetchone():
            raise Exception("Student does not exist.")
        if amount <= 6:
            raise Exception("Invalid payment amount.")
        cursor.execute(
            "INSERT INTO payments (payment_id, student_id, amount, payment_date) VALUES (%s, %s, %s, %s)",
            (payment_id, student_id, amount, payment_date)
        )
        connection.commit()
        print("Payment recorded successfully.")
    except Exception as e:
        connection.rollback()
        print("Failed to record payment. Rolled back.")
        print("Reason:", e)
    finally:
        cursor.close()
        connection.close()

if __name__ == "__main__":
    record_payment_with_transaction(payment_id=10,student_id=1,amount=100,payment_date=date.today())
```

Output

```
Payment recorded successfully.
```

	payment_id	student_id	amount	payment_date
▶	1	1	500.00	2025-06-25
	2	1	200.00	2025-06-25
	10	1	100.00	2025-06-25
*	NULL	NULL	NULL	NULL

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

```python
from util.DBConnection import DBConnection
from datetime import date, datetime

def insert_student(student_id, first_name, last_name, dob, email, phone):  1 usage
    try:
        connection = DBConnection.get_connection()
        cursor = connection.cursor()
        cursor.execute(
            "INSERT INTO students (student_id, first_name, last_name, date_of_birth, email, phone_number) "
            "VALUES (%s, %s, %s, %s, %s, %s)",
            (student_id, first_name, last_name, dob, email, phone)
        )
        connection.commit()
        print("Student inserted successfully.")
    except Exception as e:
        print("Failed to insert student:", e)
    finally:
        cursor.close()
        connection.close()

if __name__ == "__main__":
    insert_student(
        student_id=2,
        first_name="John",
        last_name="Doe",
        dob="1995-08-15",
        email="john.doe@example.com",
        phone="123-456-7890"
    )
```

Output

```
Student inserted successfully.
```

student_id	first_name	last_name	date_of_birth	email	phone_number	balance
1	Tejashree	Ganesan	2003-08-16	tejashree_updated@gmail.com	9876543210	0.00
2	John	Doe	1995-08-15	john.doe@example.com	123-456-7890	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Inserting these courses before Enrolling John

• Course 1: Introduction to Programming

• Course 2: Mathematics 101

```python
from util.DBConnection import DBConnection
from datetime import date, datetime

def insert_course(course_id, course_name, course_code, instructor_name):  2 usages
    try:
        connection = DBConnection.get_connection()
        cursor = connection.cursor()
        cursor.execute(
            "INSERT INTO courses (course_id, course_name, course_code, instructor_name) "
            "VALUES (%s, %s, %s, %s)",
            (course_id, course_name, course_code, instructor_name)
        )
        connection.commit()
        print(f"Course '{course_name}' inserted successfully.")
    except Exception as e:
        print(f"Failed to insert course '{course_name}':", e)
    finally:
        cursor.close()
        connection.close()

if __name__ == "__main__":
    insert_course(course_id=3, course_name="Introduction to Programming", course_code="CS501", instructor_name="Alice Brown")
    insert_course(course_id=4, course_name="Mathematics 101", course_code="MATH501", instructor_name="William James")
```

Output

```
Course 'Introduction to Programming' inserted successfully.
Course 'Mathematics 101' inserted successfully.
```

John is enrolling in the following courses:

• Course 1: Introduction to Programming

• Course 2: Mathematics 101

```python
from util.DBConnection import DBConnection
from datetime import date, datetime


def enroll_student(enrollment_id, student_id, course_id, enrollment_date):  2 usages
    try:
        connection = DBConnection.get_connection()
        cursor = connection.cursor()
        cursor.execute(
            "INSERT INTO enrollments (enrollment_id, student_id, course_id, enrollment_date) "
            "VALUES (%s, %s, %s, %s)",
            (enrollment_id, student_id, course_id, enrollment_date)
        )
        connection.commit()
        print(f"Enrolled in course_id {course_id} successfully.")
    except Exception as e:
        print(f"Failed to enroll in course {course_id}:", e)
    finally:
        cursor.close()
        connection.close()

if __name__ == "__main__":
    enroll_student(enrollment_id=6, student_id=2, course_id=3, enrollment_date=date.today())
    enroll_student(enrollment_id=7, student_id=2, course_id=4, enrollment_date=date.today())
```

Output

```
Enrolled in course_id 3 successfully.
Enrolled in course_id 4 successfully.
```

enrollment_id	student_id	course_id	enrollment_date
1	1	1	2025-06-25
2	1	2	2025-06-16
4	1	2	2025-06-25
5	1	2	2025-06-25
6	2	3	2025-06-25
7	2	4	2025-06-25
NULL	NULL	NULL	NULL

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

Inserting values in the teacher table first:

```python
from util.DBConnection import DBConnection
from datetime import date, datetime

def insert_teacher(teacher_id, first_name, last_name, email, expertise):  1 usage
    try:
        connection = DBConnection.get_connection()
        cursor = connection.cursor()
        cursor.execute(
            "INSERT INTO teachers (teacher_id, first_name, last_name, email, expertise) "
            "VALUES (%s, %s, %s, %s, %s)",
            (teacher_id, first_name, last_name, email, expertise)
        )
        connection.commit()
        print(f"Teacher {first_name} {last_name} inserted successfully.")
    except Exception as e:
        print("Failed to insert teacher:", e)
    finally:
        cursor.close()
        connection.close()

if __name__ == "__main__":
    insert_teacher(
        teacher_id=2,
        first_name="Sarah",
        last_name="Smith",
        email="sarah.smith@example.com",
        expertise="Computer Science"
    )
```

Output

```
Teacher Sarah Smith inserted successfully.
```

teacher_id	first_name	last_name	email	expertise
1	Olivia	Thompson	olivia@gmail.com	Computer Science
2	Sarah	Smith	sarah.smith@example.com	Computer Science
NULL	NULL	NULL	NULL	NULL

Course to be assigned:

• Course Name: Advanced Database Management

• Course Code: CS302

Inserting values in the courses table first:

```
from util.DBConnection import DBConnection
from datetime import date, datetime

def insert_course(course_id, course_name, course_code, instructor_name):  1 usage
    try:
        connection = DBConnection.get_connection()
        cursor = connection.cursor()
        cursor.execute(
            "INSERT INTO courses (course_id, course_name, course_code, instructor_name) "
            "VALUES (%s, %s, %s, %s)",
            (course_id, course_name, course_code, instructor_name)
        )
        connection.commit()
        print(f"Course '{course_name}' inserted successfully.")
    except Exception as e:
        print(f"Failed to insert course '{course_name}':", e)
    finally:
        cursor.close()
        connection.close()

if __name__ == "__main__":
    insert_course(course_id=5,course_name="Advanced Database Management",course_code="CS302",instructor_name="TBD")
```

Output

```
Course 'Advanced Database Management' inserted successfully.
```

course_id	course_name	course_code	instructor_name
1	Computer Science	CS809	William James
2	AI	CS102	Olivia Thompson
3	Introduction to Programming	CS501	Alice Brown
4	Mathematics 101	MATH501	William James
5	Advanced Database Management	CS302	TBD
NULL	NULL	NULL	NULL

The system should perform the following tasks:

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

```python
def dynamic_query_builder(table, columns="*", conditions=None, order_by=None, where=None):  1 usage
    valid_tables = {"students", "courses", "teachers", "payments", "enrollments"}
    if table not in valid_tables:
        raise ValueError("Invalid table name.")
    query = f"SELECT {columns} FROM {table}"
    params = []
    if conditions:
        condition_clauses = []
        for key, value in conditions.items():
            condition_clauses.append(f"{key} = %s")
            params.append(value)
        query += " WHERE " + " AND ".join(condition_clauses)

    if where:
        if "WHERE" in query:
            query += f" AND {where}"
        else:
            query += f" WHERE {where}"
    if order_by:
        query += f" ORDER BY {order_by}"
    try:
        connection = get_connection()
        cursor = connection.cursor()
        cursor.execute(query, params)
        results = cursor.fetchall()
        for row in results:
            print(row)
    except Exception as e:
        print("Query execution failed:", e)
    finally:
        cursor.close()
        connection.close()

print("\n Retrieve the course record from the database based on the course code:")
dynamic_query_builder( table: "courses", columns="*", where="course_code = 'CS302'")
```

Output

```
 Retrieve the course record from the database based on the course code:
(5, 'Advanced Database Management', 'CS302', 'TBD')
```

- ***Assign Sarah Smith as the instructor for the course.***
- ***Update the course record in the database with the new instructor information.***

```python
from util.DBConnection import DBConnection
from datetime import date, datetime

def assign_instructor_by_course_code(course_code, instructor_full_name, course_name):  1 usage
    try:
        connection = DBConnection.get_connection()
        cursor = connection.cursor()
        cursor.execute(
            "UPDATE courses SET instructor_name = %s WHERE course_code = %s",
            (instructor_full_name, course_code)
        )
        connection.commit()
        print(f"Instructor updated to '{instructor_full_name}' for course '{course_name}'.")
    except Exception as e:
        print("Error while updating instructor:", e)
    finally:
        cursor.close()
        connection.close()

if __name__ == "__main__":
    assign_instructor_by_course_code(course_code="CS302",instructor_full_name="Sarah Smith",course_name="Advanced Database Management")
```

Output

```
Instructor updated to 'Sarah Smith' for course 'Advanced Database Management'.
```

course_id	course_name	course_code	instructor_name
1	Computer Science	CS809	William James
2	AI	CS102	Olivia Thompson
3	Introduction to Programming	CS501	Alice Brown
4	Mathematics 101	MATH501	William James
5	Advanced Database Management	CS302	Sarah Smith
NULL	NULL	NULL	NULL

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

Inserting values in the student table first:

```python
from util.DBConnection import DBConnection
from datetime import date, datetime

def insert_student(student_id, first_name, last_name, dob, email, phone, balance):  1 usage
    try:
        connection = DBConnection.get_connection()
        cursor = connection.cursor()
        cursor.execute(
            "INSERT INTO students (student_id, first_name, last_name, date_of_birth, email, phone_number, balance) "
            "VALUES (%s, %s, %s, %s, %s, %s, %s)",
            (student_id, first_name, last_name, dob, email, phone, balance)
        )
        connection.commit()
        print("Jane Johnson inserted successfully.")
    except Exception as e:
        print("Failed to insert student:", e)
    finally:
        cursor.close()
        connection.close()

if __name__ == "__main__":
    insert_student(student_id=101,first_name="Jane",last_name="Johnson",dob=datetime.strptime( date_string: "1998-06-25",  format: "%Y-%m-%d").date(),
        email="jane.johnson@example.com",phone="999-888-7777",balance=1000.00
    )
```

Output

```
Jane Johnson inserted successfully.
```

student_id	first_name	last_name	date_of_birth	email	phone_number	balance
1	Tejashree	Ganesan	2003-08-16	tejashree_updated@gmail.com	9876543210	0.00
2	John	Doe	1995-08-15	john.doe@example.com	123-456-7890	NULL
101	Jane	Johnson	1998-06-25	jane.johnson@example.com	999-888-7777	1000.00
NULL	NULL	NULL	NULL	NULL	NULL	NULL

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.

```python
from util.DBConnection import DBConnection
from datetime import date, datetime

def record_payment(student_id, amount, payment_date):  1 usage
    try:
        connection = DBConnection.get_connection()
        cursor = connection.cursor()
        connection.start_transaction()
        print(f"Retrieving student with ID {student_id}")
        cursor.execute("SELECT * FROM students WHERE student_id = %s", (student_id,))
        student = cursor.fetchone()
        if not student:
            raise Exception("Student not found.")
        print("Student record retrieved:", student)
        print("\nInserting payment into 'payments' table")
        cursor.execute(
            "INSERT INTO payments (payment_id, student_id, amount, payment_date) "
            "VALUES (%s, %s, %s, %s)",
            (11, student_id, amount, payment_date)
        )
        print("Payment inserted successfully.")
        print("\nUpdating student's balance")
        cursor.execute(
            "UPDATE students SET balance = balance + %s WHERE student_id = %s",
            (amount, student_id)
        )
        print("Student balance updated successfully.")
        connection.commit()
        print("\nPayment recorded and balance updated successfully. Transaction committed.")
    except Exception as e:
        connection.rollback()
        print("\nFailed to record payment. Rolled back transaction.")
        print("Reason:", e)
    finally:
        cursor.close()
        connection.close()

if __name__ == "__main__":
    record_payment(student_id=101,amount=500.00,payment_date=datetime.strptime( date_string: "2023-04-10",  format: "%Y-%m-%d").date())
```

Output

```
Retrieving student with ID 101
Student record retrieved: (101, 'Jane', 'Johnson', datetime.date(1998, 6, 25), 'jane.johnson@example.com', '999-888-7777', Decimal('1000.00'))

Inserting payment into 'payments' table
Payment inserted successfully.

Updating student's balance
Student balance updated successfully.

Payment recorded and balance updated successfully. Transaction committed.
```

payment_id	student_id	amount	payment_date
1	1	500.00	2025-06-25
2	1	200.00	2025-06-25
10	1	100.00	2025-06-25
11	101	500.00	2023-04-10
NULL	NULL	NULL	NULL

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 10

Inserting values in the courses table first:

```python
from util.DBConnection import DBConnection
from datetime import date, datetime

def insert_course():  1 usage
    try:
        connection = DBConnection.get_connection()
        cursor = connection.cursor()
        cursor.execute(
            "INSERT INTO courses (course_id, course_name, course_code, instructor_name) "
            "VALUES (%s, %s, %s, %s)",
            (6, "Computer Science 101", "CS101", "Dr. Alan Turing")
        )
        connection.commit()
        print("Course 'Computer Science 101' inserted successfully.")
    except Exception as e:
        print("Failed to insert course:", e)
    finally:
        cursor.close()
        connection.close()

if __name__ == "__main__":
    insert_course()
```

Output

```
Course 'Computer Science 101' inserted successfully.
```

course_id	course_name	course_code	instructor_name
1	Computer Science	CS809	William James
2	AI	CS102	Olivia Thompson
3	Introduction to Programming	CS501	Alice Brown
4	Mathematics 101	MATH501	William James
5	Advanced Database Management	CS302	Sarah Smith
6	Computer Science 101	CS101	Dr. Alan Turing

Then, Adding Enrollments for the particular course

```python
from util.DBConnection import DBConnection
from datetime import date, datetime


def enroll_students():  1 usage
    try:
        connection = DBConnection.get_connection()
        cursor = connection.cursor()
        enrollments = [
            (8, 1, 6, date.today()),
            (9, 2, 6, date.today())
        ]
        for enrollment in enrollments:
            cursor.execute(
                "INSERT INTO enrollments (enrollment_id, student_id, course_id, enrollment_date) "
                "VALUES (%s, %s, %s, %s)",
                enrollment
            )
        connection.commit()
        print("Two students enrolled in 'Computer Science 101'.")
    except Exception as e:
        connection.rollback()
        print("Enrollment failed:", e)
    finally:
        cursor.close()
        connection.close()

if __name__ == "__main__":
    enroll_students()
```

Output

```
Two students enrolled in 'Computer Science 101'.
```

enrollment_id	student_id	course_id	enrollment_date
1	1	1	2025-06-25
2	1	2	2025-06-16
4	1	2	2025-06-25
5	1	2	2025-06-25
6	2	3	2025-06-25
7	2	4	2025-06-25
8	1	6	2025-06-25
9	2	6	2025-06-25
NULL	NULL	NULL	NULL

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

```python
from util.DBConnection import DBConnection
from datetime import date, datetime

def generate_enrollment_report(course_name, course_id):  1 usage
    try:
        connection = DBConnection.get_connection()
        cursor = connection.cursor()
        cursor.execute("SELECT course_id FROM courses WHERE course_name = %s", (course_name,))
        course_result = cursor.fetchone()

        cursor.execute(
            "SELECT s.student_id, s.first_name, s.last_name, s.email, s.phone_number, e.enrollment_date "
            "FROM enrollments e "
            "JOIN students s ON e.student_id = s.student_id "
            "WHERE e.course_id = %s",
            (course_id,)
        )
        enrollments = cursor.fetchall()

        if not enrollments:
            print("No students enrolled in this course.")
            return

        print(f"\nEnrollment Report for {course_name}:")
        print("-----------------------------------------------------")
        for student in enrollments:
            print(f"Student ID: {student[0]}, Name: {student[1]} {student[2]}, Email: {student[3]}, Phone: {student[4]}, Enrolled On: {student[5]}")
        print("-----------------------------------------------------")
        print(f"Total Students Enrolled: {len(enrollments)}")
    except Exception as e:
        print("Failed to generate report:", e)
    finally:
        cursor.close()
        connection.close()

if __name__ == "__main__":
    generate_enrollment_report(course_name="Computer Science 101", course_id=6)
```

Output

```
Enrollment Report for 'Computer Science 101':
-----------------------------------------------------
Student ID: 1, Name: Tejashree Ganesan, Email: tejashree_updated@gmail.com, Phone: 9876543210, Enrolled On: 2025-06-25
Student ID: 2, Name: John Doe, Email: john.doe@example.com, Phone: 123-456-7890, Enrolled On: 2025-06-25
-----------------------------------------------------
Total Students Enrolled: 2
```