Hexaware Foundation Training

Python Assignment

STUDENT INFORMATION SYSTEM

NAME: Aathirainathan P

DATE: 05-10-2024

Task 1: Define Classes

1.1 entity/student:

```
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

```

1.2 entity/course

```
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

```

1.3 entity/enrollment

```
class Enrollment:
    def __init__(self, enrollment_id, student_id, course_id, enrollment_date):
        self.enrollment_id = enrollment_id
        self.student_id = student_id
        self.course_id = course_id
        self.enrollment_date = enrollment_date

```

1.4 entity/teacher

```
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

```

1.5 entity/payment

```
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

```

1.6 SIS Class

```
class SIS:
    def __init__(self):
        self.students = []       # List to hold all students
        self.courses = []        # List to hold all courses
        self.teachers = []       # List to hold all teachers
        self.enrollments = []    # List to hold all enrollments
        self.payments = []       # List to hold all payments
```

Task 2: Implement Constructors

Already implemented in Task 1.

2.1 entity/student

```
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

```

2.2 entity/course

```
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

```

2.3 entity/enrollment

```
class Enrollment:
    def __init__(self, enrollment_id, student_id, course_id, enrollment_date):
        self.enrollment_id = enrollment_id
        self.student_id = student_id
        self.course_id = course_id
        self.enrollment_date = enrollment_date

```

2.4 entity/teacher

```
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

```

2.5 entity/payment

```
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

```

2.6 entity/ SIS

```
class SIS:
    def __init__(self):
        self.students = []       # List to hold all students
        self.courses = []        # List to hold all courses
        self.teachers = []       # List to hold all teachers
        self.enrollments = []    # List to hold all enrollments
        self.payments = []       # List to hold all payments
```

Task 3: Implement Methods

3.1 Student Class

```python
import sys
import os

base_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), ".."))
sys.path.append(base_dir)

from entity.payment import Payment

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
        self.enrolled_courses = []  # List to store enrolled courses
        self.payments = []           # List to store payment records

    def enroll_in_course(self, course):
        self.enrolled_courses.append(course)

    def update_student_info(self, first_name, last_name, date_of_birth, email, phone_number):
        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

    def make_payment(self, amount, payment_date):
        payment = Payment(None, self.student_id, amount, payment_date)
        self.payments.append(payment)

    def display_student_info(self):
        print(f"Student ID: {self.student_id}")
        print(f"Name: {self.first_name} {self.last_name}")
        print(f"Date of Birth: {self.date_of_birth}")
        print(f"Email: {self.email}")
        print(f"Phone Number: {self.phone_number}")

    def get_enrolled_courses(self):
        return self.enrolled_courses

    def get_payment_history(self):
        return self.payments
```

3.2 Course Class:

```
class Course:
    def __init__(self, course_id, course_name, course_code, instructor_name):
        self.course_id = course_id
        self.course_name = course_name
        self.course_code = course_code
        self.instructor_name = instructor_name
        self.enrollments = []  # List to store enrollments
        self.teacher = None

    def assign_teacher(self, teacher):
        self.teacher = teacher

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

    def display_course_info(self):
        print(f"Course ID: {self.course_id}")
        print(f"Course Name: {self.course_name}")
        print(f"Course Code: {self.course_code}")
        print(f"Instructor Name: {self.instructor_name}")

    def get_enrollments(self):
        return self.enrollments

    def get_teacher(self):
        return self.teacher
```

3.3 Enrollment class:

```
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

    def get_student_id(self):
        return self.student_id

    def get_course_id(self):
        return self.course_id
```

3.4 Teacher Class:

```
class Teacher:
    def __init__(self, teacher_id, first_name, last_name, email):
        self.teacher_id = teacher_id
        self.first_name = first_name
```

```
        self.last_name = last_name
        self.email = email
        self.assigned_courses = []  # List to store assigned courses

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

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

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

3.5 Payment Class:

```
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

    def get_student(self):
        return self.student_id

    def get_payment_amount(self):
        return self.amount

    def get_payment_date(self):
        return self.payment_date

```

3.6 SIS Class:

```
import sys
import os

base_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), ".."))
sys.path.append(base_dir)

from entity.enrollment import Enrollment
from entity.payment import Payment
from entity.student import Student

class SIS:
    def __init__(self):
        self.students = []
        self.courses = []
```

```python
        self.enrollments = []
        self.payments = []

    def enroll_student_in_course(self, student, course):
        enrollment_date = '2024-01-01'
        enrollment = Enrollment(len(self.enrollments) + 1, student.student_id, course.course_id, enrollment_date)
        self.enrollments.append(enrollment)
        print(f'Enrolled {student.first_name} in {course.course_name} on {enrollment_date}')

    def record_payment(self, student, amount, payment_date):
        payment = Payment(len(self.payments) + 1, student.student_id, amount, payment_date)
        self.payments.append(payment)
        print(f'Recorded payment of {amount} from {student.first_name} on {payment_date}')

    def generate_enrollment_report(self, course):
        enrollments = [enrollment for enrollment in self.enrollments if enrollment.course_id == course.course_id]
        if not enrollments:
            print(f"\nNo enrollments found for {course.course_name}")
            return enrollments

        print(f'\nEnrollment Report for {course.course_name}:')
        for enrollment in enrollments:
            print(f'Student ID: {enrollment.student_id}, Course ID: {enrollment.course_id}, Enrollment Date: {enrollment.enrollment_date}')

        return enrollments

    def generate_payment_report(self, student):
        payment_report = [payment for payment in self.payments if payment.student_id == student.student_id]
        print(f'\nPayment Report for {student.first_name} {student.last_name}:')
        for payment in payment_report:
            print(f'Amount: {payment.amount}, Payment Date: {payment.payment_date}')

    def calculate_course_statistics(self, course):
        enrollments = [enrollment for enrollment in self.enrollments if enrollment.course_id == course.course_id]
        if not enrollments:
            print(f"\nNo enrollments found for {course.course_name}")
            return 0, 0
        total_payments = sum(payment.amount for payment in self.payments if payment.student_id in [e.student_id for e in enrollments])
        return len(enrollments), total_payments
```

3.7 Testing the implemented methods:

```python
import sys
import os

base_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), ".."))
sys.path.append(base_dir)

from entity.student import Student
from entity.course import Course
from entity.teacher import Teacher
from entity.enrollment import Enrollment
from entity.payment import Payment
from entity.sis import SIS

def main():
    sis = SIS()

    student1 = Student(1, 'John', 'Doe', '1995-08-15', 'john.doe@example.com', '1234567890')
    student2 = Student(2, 'Jane', 'Smith', '1996-09-25', 'jane.smith@example.com', '0987654321')

    course1 = Course(1, 'Mathematics', 'MATH101', 'Dr. Alice')
    course2 = Course(2, 'Physics', 'PHYS101', 'Dr. Bob')

    sis.students.append(student1)
    sis.students.append(student2)
    sis.courses.append(course1)
    sis.courses.append(course2)

    sis.enroll_student_in_course(student1, course1)
    sis.enroll_student_in_course(student2, course2)

    print()

    sis.record_payment(student1, 5000.00, '2024-01-20')
    sis.record_payment(student2, 6000.00, '2024-02-15')

    print()
    sis.generate_enrollment_report(course2)
    print()

    sis.generate_payment_report(student1)

    num_enrollments, total_payments = sis.calculate_course_statistics(course1)
    print(f'\nStatistics for {course1.course_name}:')
    print(f'Number of Enrollments: {num_enrollments}, Total Payments: {total_payments}')

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

Output:

```
PS C:\Users\pumak\OneDrive\Desktop\Student Information System> & C:/Users/pumak/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/pumak/OneDrive/Desktop/St
udent Information System/entity/tester.py"
Enrolled John in Mathematics on 2024-01-01
Enrolled Jane in Physics on 2024-01-01
Enrolled Jane in Physics on 2024-01-01

Recorded payment of 5000.0 from John on 2024-01-20
Enrolled Jane in Physics on 2024-01-01

Enrolled Jane in Physics on 2024-01-01

Recorded payment of 6000.0 from Jane on 2024-02-15


Enrollment Report for Physics:
Student ID: 2, Course ID: 2, Enrollment Date: 2024-01-01


Payment Report for John Doe:
Amount: 5000.0, Payment Date: 2024-01-20

Statistics for Mathematics:
Number of Enrollments: 1, Total Payments: 5000.0
PS C:\Users\pumak\OneDrive\Desktop\Student Information System>
```

Task 4: Exceptions handling and Custom Exceptions

Exception/custom_exceptions.py

All the given custom exceptions are implemented below.

```python
import sys
import os

base_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), ".."))
sys.path.append(base_dir)

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

class CourseNotFoundException(Exception):
    def __init__(self, message="Course not found in the system."):
        self.message = message
        super().__init__(self.message)

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

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

class PaymentValidationException(Exception):
    def __init__(self, message="Payment validation failed."):
        self.message = message
        super().__init__(self.message)
```

```python

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

class InvalidCourseDataException(Exception):
    def __init__(self, message="Invalid data provided for the course."):
        self.message = message
        super().__init__(self.message)

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

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

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

```

Task 5: Collections

Implement Collections:

5.1 Student Class:

Two lists are created here namely enrolled_courses and payments, to list the couses enrolled by a student and payment respectively.

```python
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
        self.enrolled_courses = []  # List to store enrolled courses
        self.payments = []           # List to store payment records

    def enroll_in_course(self, course):
        self.enrolled_courses.append(course)

    def get_enrolled_courses(self):
        return self.enrolled_courses
```

5.2 Course class:

A list to store enrollments has been created with getters and setters namely, get_enrollments and enroll_student.

```
class Course:
    def __init__(self, course_id, course_name, course_code, instructor_name):
        self.course_id = course_id
        self.course_name = course_name
        self.course_code = course_code
        self.instructor_name = instructor_name
        self.enrollments = []  # List to store Enrollment objects
        self.teacher = None

    def enroll_student(self, student, enrollment_date):
        enrollment = Enrollment(len(self.enrollments) + 1, student.student_id,
   self.course_id, enrollment_date)
        self.enrollments.append(enrollment)

    def get_enrollments(self):
        return self.enrollments

```

5.3 Enrollment Class:

References for student and course has been created along with getters and setters.

```
class Enrollment:
    def __init__(self, enrollment_id, student, course, enrollment_date):
        self.enrollment_id = enrollment_id
        self.student_id = None
        self.course_id = None
        self.enrollment_date = enrollment_date
        self.student = student # To hold reference to Student object
        self.course = course   # To hold reference to Course object

    def set_student(self, student):
        self.student = student  # Method to set the Student reference

    def get_student(self):
        return self.student  # Method to get the Student object

    def set_course(self, course):
        self.course = course  # Method to set the Course reference

    def get_course(self):
        return self.course  # Method to get the Course object

    def display_enrollment_info(self):
        student_name = f"{self.student.first_name} {self.student.last_name}" if
self.student else "N/A"
        course_name = self.course.course_name if self.course else "N/A"
```

```
        print(f"Enrollment ID: {self.enrollment_id}")
        print(f"Student Name: = (Name: {student_name})")
        print(f"Course Name:(Course Name: {course_name})")
        print(f"Enrollment Date: {self.enrollment_date}")
```

5.4 Teacher Class:

A list for assigned_courses has been created.

```
class Teacher:
    def __init__(self, teacher_id, first_name, last_name, email):
        self.teacher_id = teacher_id
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.assigned_courses = []  # List to store assigned courses

    def assign_course(self, course):
        if course not in self.assigned_courses:
            self.assigned_courses.append(course)
            course.assign_teacher(self)  # Assuming Course has an assign_teacher method
        else:
            print(f"{self.first_name} is already assigned to {course.course_name}.")

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

5.5 Payment Class:

A student reference to the payment class has been created. `class Payment:`

```
    def __init__(self, payment_id, student, amount, payment_date):
        self.payment_id = payment_id
        self.student = student; # Store reference to Student object
        self.student_id = None # Store student ID
        self.amount = amount
        self.payment_date = payment_date

    def get_student(self):
        return self.student  # Return the Student object

    def set_student(self, student):
        self.student = student  # Method to set the Student referen

    def get_student_id(self):
        return self.student_id  # Return the Student ID

    def get_payment_amount(self):
        return self.amount  # Return the payment amount

    def get_payment_date(self):
```

```
        return self.payment_date  # Return the payment date

    def display_payment_info(self):
        print(f"Payment ID: {self.payment_id}")
        print(f"Student ID: {self.student_id} (Name: {self.student.first_name}
{self.student.last_name})")
        print(f"Amount: {self.amount}")
        print(f"Payment Date: {self.payment_date}")
```

Task 6: Create Methods for Managing Relationships

6.1 AddEnrollment:

```
    def AddEnrollment(self, student, course, enrollment_date):

        if any(enrollment.student_id == student.student_id and enrollment.course_id ==
course.course_id for enrollment in self.enrollments):
            raise Exception("Student is already enrolled in this course.")

        enrollment = Enrollment(len(self.enrollments) + 1, student.student_id,
course.course_id, enrollment_date)

        enrollment.set_student(student)
        enrollment.set_course(course)

        self.enrollments.append(enrollment)

        student.enrolled_courses.append(course)
        course.enrollments.append(enrollment)

        print(f'Enrolled {student.first_name} in {course.course_name} on
{enrollment_date}')
```

6.2 AssignCourseToTeacher:

```
    def assign_course_to_teacher(self, course, teacher):
        if course not in teacher.assigned_courses:
            teacher.assigned_courses.append(course)
            print(f'Course {course.course_name} has been assigned to teacher
{teacher.first_name} {teacher.last_name}.')
        else:
            print(f'Course {course.course_name} is already assigned to teacher
{teacher.first_name} {teacher.last_name}.')

```

6.3 AddPayment:

```
def add_payment(self, student, amount, payment_date):

        payment = Payment(len(self.payments) + 1, student.student_id, amount,
payment_date)
```

```python

        student.make_payment(amount, payment_date)

        self.payments.append(payment)

        print(f'Recorded payment of {amount} from {student.first_name} on {payment_date}')
```

6.4 GetEnrollmentsForStudent:

```python
def GetEnrollmentsForStudent(self, student):
        return [enrollment for enrollment in self.enrollments if enrollment.student.student_id == student.student_id]

```

6.5 GetCoursesForTeacher:

```python
    def GetCoursesForTeacher(self, teacher):
        return [course for course in self.courses if course in teacher.assigned_courses]

```

6.6 Main Method:

```python
import sys
import os

base_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), ".."))
sys.path.append(base_dir)

from entity.student import Student
from entity.course import Course
from entity.enrollment import Enrollment
from entity.sis import SIS
from entity.teacher import Teacher
from entity.payment import Payment
from dao.sisserviceprovider import SISServiceProvider
from exception.custom_exceptions import (
    DuplicateEnrollmentException,
    CourseNotFoundException,
    StudentNotFoundException,
    TeacherNotFoundException,
    PaymentValidationException,
    InvalidStudentDataException,
    InvalidCourseDataException,
    InvalidEnrollmentDataException,
    InvalidTeacherDataException,
    InsufficientFundsException
)

def main():
```

```python
    sis = SIS()

    student1 = Student(1, 'John', 'Doe', '1995-08-15', 'john.doe@example.com',
'1234567890')
    student2 = Student(2, 'Jane', 'Smith', '1996-09-25', 'jane.smith@example.com',
'0987654321')
    student3 = Student(3, 'Tom', 'Brown', '1997-11-30', 'tom.brown@example.com',
'1122334455')

    course1 = Course(1, 'Mathematics', 'MATH101', 'Dr. Alice')
    course2 = Course(2, 'Physics', 'PHYS101', 'Dr. Bob')
    course3 = Course(3, 'Chemistry', 'CHEM101', 'Dr. Carol')

    teacher1 = Teacher(1, 'Alice', 'Johnson', 'alice.johnson@example.com')
    teacher2 = Teacher(2, 'Bob', 'Smith', 'bob.smith@example.com')
    teacher3 = Teacher(3, 'Carol', 'White', 'carol.white@example.com')

    sis.students.append(student1)
    sis.students.append(student2)
    sis.students.append(student3)

    sis.courses.append(course1)
    sis.courses.append(course2)
    sis.courses.append(course3)

    sis.teachers.append(teacher1)
    sis.teachers.append(teacher2)
    sis.teachers.append(teacher3)

    try:
        sis.AddEnrollment(student1, course1, '2024-01-01')
        sis.AddEnrollment(student1, course1, '2024-01-01')
    except DuplicateEnrollmentException as e:
        print(f"Error: {e}")

    try:
        sis.AddEnrollment(student2, course2, '2024-01-01')
        sis.AddEnrollment(student3, course3, '2024-01-02')
    except Exception as e:
        print(f"Error: {e}")

    print()

    try:
        sis.assign_course_to_teacher(course1, teacher1)
        sis.assign_course_to_teacher(course1, teacher1)
        sis.assign_course_to_teacher(course3, teacher3)
    except Exception as e:
        print(f"Error: {e}")

    print()

    try:
        sis.add_payment(student1, 5000, '2024-01-20')
```

```
        sis.add_payment(student2, 6000, '2024-02-15')
    except Exception as e:
        print(f"Error: {e}")

    print()
    try:
        enrollments_for_john = sis.GetEnrollmentsForStudent(student1)
        print(f"\nEnrollments for {student1.first_name} {student1.last_name}:")
        for enrollment in enrollments_for_john:
            print(f'Enrolled in {enrollment.course.course_name} on
{enrollment.enrollment_date}')
    except Exception as e:
        print(f"Error: {e}")
    print()

    try:
        courses_for_alice = sis.GetCoursesForTeacher(teacher1)
        print(f"\nCourses assigned to {teacher1.first_name} {teacher1.last_name}:")
        for course in courses_for_alice:
            print(course.course_name)
    except Exception as e:
        print(f"Error: {e}")
    print()

    try:
        non_existent_student = Student(4, 'Mike', 'Jones', '1998-12-01',
'mike.jones@example.com', '9999999999')
        sis.AddEnrollment(non_existent_student, course1, '2024-01-01')
    except StudentNotFoundException as e:
        print(f"Error: {e}")
    print()

    try:
        non_existent_teacher = Teacher(4, 'Nina', 'Green', 'nina.green@example.com')
        sis.assign_course_to_teacher(course1, non_existent_teacher)
    except TeacherNotFoundException as e:
        print(f"Error: {e}")



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

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS                                                    Python

PS C:\Users\pumak\OneDrive\Desktop\Student Information System> & C:/Users/pumak/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/pumak/OneDrive/Desktop/St
udent Information System/main/main_module.py"
Enrolled John in Mathematics on 2024-01-01
Enrolled John in Mathematics on 2024-01-01
Enrolled Jane in Physics on 2024-01-01
Enrolled Tom in Chemistry on 2024-01-02

Course Mathematics has been assigned to teacher Alice Johnson.
Course Mathematics is already assigned to teacher Alice Johnson.
Course Chemistry has been assigned to teacher Carol White.

Recorded payment of 5000 from John on 2024-01-20
Recorded payment of 6000 from Jane on 2024-02-15


Enrollments for John Doe:
Enrolled in Mathematics on 2024-01-01
Enrolled in Mathematics on 2024-01-01


Courses assigned to Alice Johnson:
Mathematics

Enrolled Mike in Mathematics on 2024-01-01

Course Mathematics has been assigned to teacher Nina Green.
PS C:\Users\pumak\OneDrive\Desktop\Student Information System>
```

Task 7: Database Connectivity

7.1 Database Initialization:

7.1.1 Util/db_property_util:

```python
class DBPropertyUtil:
    @staticmethod
    def get_connection_string():
        return 'Driver={SQL Server};Server=PUMA\\SQLEXPRESS;Database=SISDB;Trusted_Connection=yes;'


```

7.1.2 util/db_conn_util:

```python
# DBConnUtil.py
import sys
import os

base_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), ".."))
sys.path.append(base_dir)

import pyodbc
from util.db_property_util import DBPropertyUtil

class DBConnUtil:
    @staticmethod
    def get_connection():
        connection_string = DBPropertyUtil.get_connection_string()
        try:
            conn = pyodbc.connect(connection_string)
            print("Connected Successfully")
```

```
            return conn
        except Exception as e:
            print("Connection failed:", e)
            return None


```

7.1. util/DatabaseManager(For DB Initialization):

```
import sys
import os

base_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), ".."))
sys.path.append(base_dir)
import pyodbc
from util.db_conn_util import DBConnUtil

class DatabaseManager:
    def __init__(self):
        self.conn = DBConnUtil.get_connection()
        if self.conn:
            self.cursor = self.conn.cursor()
            self.initialize_database()

    def initialize_database(self):
        sql_commands = [
            '''
            IF NOT EXISTS (SELECT * FROM sysobjects WHERE name='Students' AND xtype='U')
            CREATE TABLE Students (
                id INT PRIMARY KEY IDENTITY(1,1),
                first_name NVARCHAR(100) NOT NULL,
                last_name NVARCHAR(100) NOT NULL,
                dob DATE NOT NULL,
                email NVARCHAR(255) NOT NULL UNIQUE,
                phone NVARCHAR(15) NOT NULL
            );
            ''',
            '''
            IF NOT EXISTS (SELECT * FROM sysobjects WHERE name='Courses' AND xtype='U')
            CREATE TABLE Courses (
                id INT PRIMARY KEY IDENTITY(1,1),
                course_name NVARCHAR(100) NOT NULL,
                course_code NVARCHAR(20) NOT NULL UNIQUE,
                instructor_name NVARCHAR(100) NOT NULL
            );
            ''',
            '''
            IF NOT EXISTS (SELECT * FROM sysobjects WHERE name='Enrollments' AND xtype='U')
            CREATE TABLE Enrollments (
                id INT PRIMARY KEY IDENTITY(1,1),
                student_id INT NOT NULL,
```

```
                course_id INT NOT NULL,
                enrollment_date DATE NOT NULL,
                FOREIGN KEY (student_id) REFERENCES Students (id),
                FOREIGN KEY (course_id) REFERENCES Courses (id),
                UNIQUE (student_id, course_id)
            );
            ''',
            '''
            IF NOT EXISTS (SELECT * FROM sysobjects WHERE name='Teachers' AND xtype='U')
            CREATE TABLE Teachers (
                id INT PRIMARY KEY IDENTITY(1,1),
                first_name NVARCHAR(100) NOT NULL,
                last_name NVARCHAR(100) NOT NULL,
                email NVARCHAR(255) NOT NULL UNIQUE
            );
            ''',
            '''
            IF NOT EXISTS (SELECT * FROM sysobjects WHERE name='Payments' AND xtype='U')
            CREATE TABLE Payments (
                id INT PRIMARY KEY IDENTITY(1,1),
                student_id INT NOT NULL,
                amount DECIMAL(10, 2) NOT NULL,
                payment_date DATE NOT NULL,
                FOREIGN KEY (student_id) REFERENCES Students (id)
            );
            '''
        ]

        for command in sql_commands:
            self.cursor.execute(command)

        self.conn.commit()

    def close(self):
        if self.conn:
            self.conn.close()


```

7.2 Data Retrieval:

```
def dynamic_query(self, table, columns=None, conditions=None, order_by=None):
        try:
            columns = ', '.join(columns) if columns else '*'
            query = f"SELECT {columns} FROM {table}"
            if conditions:
                query += " WHERE " + ' AND '.join(conditions)
            if order_by:
                query += " ORDER BY " + order_by

            self.cursor.execute(query)
```

```
            return self.cursor.fetchall()
        except Exception as e:
            print("Error executing dynamic query:", e)
            return []

    def get_students(self):
        return self.dynamic_query("Students")

    def get_courses(self):
        return self.dynamic_query("Courses")

    def get_enrollments(self):
        return self.dynamic_query("Enrollments")

    def get_teachers(self):
        return self.dynamic_query("Teachers")

    def get_payments(self):
        return self.dynamic_query("Payments")

```

7.3 Data Insertion and Updating:

```
def insert_student(self, first_name, last_name, dob, email, phone):
        try:
            self.cursor.execute(
                "INSERT INTO Students (first_name, last_name, dob, email, phone) VALUES (?, ?, ?, ?, ?)",
                (first_name, last_name, dob, email, phone)
            )
            self.conn.commit()
            print(f"Inserted student: {first_name} {last_name}")
        except Exception as e:
            print("Error inserting student:", e)

    def update_student(self, student_id, first_name, last_name, dob, email, phone):
        try:
            self.cursor.execute(
                "UPDATE Students SET first_name = ?, last_name = ?, dob = ?, email = ?, phone = ? WHERE id = ?",
                (first_name, last_name, dob, email, phone, student_id)
            )
            self.conn.commit()
            print(f"Updated student ID {student_id}")
        except Exception as e:
            print("Error updating student:", e)

    def insert_enrollment(self, student_id, course_id, enrollment_date):
        try:
            self.cursor.execute(
                "INSERT INTO Enrollments (student_id, course_id, enrollment_date) VALUES (?, ?, ?)",
                (student_id, course_id, enrollment_date)
```

```
            )
            self.conn.commit()
            print(f"Inserted enrollment for student ID {student_id} in course ID {course_id}")
        except Exception as e:
            print("Error inserting enrollment:", e)

    def record_payment(self, student_id, amount, payment_date):
        try:
            self.cursor.execute(
                "INSERT INTO Payments (student_id, amount, payment_date) VALUES (?, ?, ?)",
                (student_id, amount, payment_date)
            )
            self.conn.commit()
            print(f"Inserted payment of {amount} from student ID {student_id}")
        except Exception as e:
            print("Error inserting payment:", e)

```

7.4 Transaction Management:

```
def begin_transaction(self):
        self.conn.autocommit = False

    def commit_transaction(self):
        self.conn.commit()
        self.conn.autocommit = True

    def rollback_transaction(self):
        self.conn.rollback()
        self.conn.autocommit = True

def insert_student(self, first_name, last_name, dob, email, phone):
        try:
            self.cursor.execute(
                "INSERT INTO Students (first_name, last_name, dob, email, phone) VALUES (?, ?, ?, ?, ?)",
                (first_name, last_name, dob, email, phone)
            )
            self.conn.commit()
            print(f"Inserted student: {first_name} {last_name}")
        except Exception as e:
            print("Error inserting student:", e)

    def update_student(self, student_id, first_name, last_name, dob, email, phone):
        try:
            self.cursor.execute(
                "UPDATE Students SET first_name = ?, last_name = ?, dob = ?, email = ?, phone = ? WHERE id = ?",
                (first_name, last_name, dob, email, phone, student_id)
            )
            self.conn.commit()
```

```
            print(f"Updated student ID {student_id}")
        except Exception as e:
            print("Error updating student:", e)

    def insert_enrollment(self, student_id, course_id, enrollment_date):
        try:
            self.cursor.execute(
                "INSERT INTO Enrollments (student_id, course_id, enrollment_date) VALUES (?, ?, ?)",
                (student_id, course_id, enrollment_date)
            )
            self.conn.commit()
            print(f"Inserted enrollment for student ID {student_id} in course ID {course_id}")
        except Exception as e:
            print("Error inserting enrollment:", e)

    def record_payment(self, student_id, amount, payment_date):
        try:
            self.cursor.execute(
                "INSERT INTO Payments (student_id, amount, payment_date) VALUES (?, ?, ?)",
                (student_id, amount, payment_date)
            )
            self.conn.commit()
            print(f"Inserted payment of {amount} from student ID {student_id}")
        except Exception as e:
            print("Error inserting payment:", e)

    def enroll_student_with_payment(self, student_id, course_id, enrollment_date, amount, payment_date):
        self.begin_transaction()
        try:
            self.insert_enrollment(student_id, course_id, enrollment_date)
            self.record_payment(student_id, amount, payment_date)
            self.commit_transaction()
        except Exception as e:
            self.rollback_transaction()
            print("Transaction failed:", e)


```

7.5 Dynamic Query Builder:

```
def dynamic_query(self, table, columns=None, conditions=None, order_by=None):
        try:
            columns = ', '.join(columns) if columns else '*'
            query = f"SELECT {columns} FROM {table}"
            if conditions:
                query += " WHERE " + ' AND '.join(conditions)
            if order_by:
                query += " ORDER BY " + order_by

```

```
            self.cursor.execute(query)
            return self.cursor.fetchall()
        except Exception as e:
            print("Error executing dynamic query:", e)
            return []

```

Task 8: Student Enrollment:

```
import sys
import os

base_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), ".."))
sys.path.append(base_dir)

from util.DatabaseManager import DatabaseManager
from util.db_conn_util import DBConnUtil
from entity.student import Student
from entity.course import Course
from entity.enrollment import Enrollment
from entity.sis import SIS
from entity.teacher import Teacher
from entity.payment import Payment
from exception.custom_exceptions import (
    DuplicateEnrollmentException,
    CourseNotFoundException,
    StudentNotFoundException,
    TeacherNotFoundException,
    PaymentValidationException,
    InvalidStudentDataException,
    InvalidCourseDataException,
    InvalidEnrollmentDataException,
    InvalidTeacherDataException,
    InsufficientFundsException
)

if __name__ == "__main__":
    db_manager = DatabaseManager()

    first_name = 'John'
    last_name = 'Doe'
    dob = '1995-08-15'
    email = 'john.doe@example.com'
    phone = '123-456-7890'

    db_manager.insert_student(first_name, last_name, dob, email, phone)

    course_names = ['Introduction to Programming', 'Mathematics 101']
    course_ids = []

    for course_name in course_names:

```

```
        course = db_manager.dynamic_query(
            "Courses",
            columns=["course_id"],
            conditions=[f"course_name = '{course_name}'"]
        )
        if course:
            course_ids.append(course[0][0])

    student_id = db_manager.dynamic_query("Students", columns=["student_id"], conditions=[f"email = '{email}'"])[0][0]

    enrollment_date = "2024-01-01”
    for course_id in course_ids:
        db_manager.insert_enrollment(student_id, course_id, enrollment_date)

    print(f"John Doe has been enrolled in the following courses: {course_names}")


    db_manager.close()

```

Output:

```
PS> & C:/Users/pumak/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/pumak/OneDrive/Desktop/Student Information System/main/task8main.py"
Connected Successfully
Inserted student: John Doe
John Doe has been enrolled in the following courses: ['Introduction to Programming', 'Mathematics 101']
PS C:\Users\pumak\OneDrive\Desktop\Student Information System>
```

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

```
import sys
import os

base_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), ".."))
sys.path.append(base_dir)
import pyodbc
from util.db_conn_util import DBConnUtil
from util.DatabaseManager import DatabaseManager


if __name__ == "__main__":
    db_manager = DatabaseManager()


    first_name = 'Sarah'
    last_name = 'Smith'
    email = 'sarah.smith@example.com'

    db_manager.insert_teacher(first_name, last_name, email)
```

```python
    course_id = 12


    query = "SELECT * FROM Courses WHERE course_id = ?"
    course = db_manager.execute_query(query, (course_id,))

    if course:
        teacher = db_manager.get_teacher_by_email(email)

        if teacher:

            update_query = "UPDATE Courses SET teacher_id = ? WHERE course_id = ?"
            db_manager.execute_query(update_query, (teacher[0], course_id))
            print(f"Assigned {first_name} {last_name} to teach course ID {course_id}.")
        else:
            print("Teacher not found.")
    else:
        print("Course not found.")

    db_manager.close()
```

Output:

```
PS C:\Users\pumak\OneDrive\Desktop\Student Information System> & C:/Users/pumak/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/
eDrive/Desktop/Student Information System/main/task9main.py"
Connected Successfully
Inserted teacher: Sarah Smith
Assigned Sarah Smith to teach course ID 12.
PS C:\Users\pumak\OneDrive\Desktop\Student Information System>
```

Updated teacher and course databases:

100 %

Results | Messages

	teacher_id	first_name	last_name	email
1	2	Ram	Narayan	ram.narayan@example.com
2	3	Janani	Sivakumar	janani.sivakumar@example.com
3	4	Saravanan	Raja	saravanan.raja@example.com
4	5	Muthu	Palanisamy	muthu.palanisamy@example.com
5	6	Vijaya	Lakshmi	vijaya.lakshmi@example.com
6	7	Kamal	Mani	kamal.mani@example.com
7	8	Radhika	Sankar	radhika.sankar@example.com
8	9	Ganesh	Perumal	ganesh.perumal@example.com
9	10	Thiru	Arasan	thiru.arasan@example.com
10	11	Madhavi	Natesan	madhavi.natesan@example.com
11	12	kumar	sangakara	ks@gmail.com
12	19	Sarah	Smith	sarah.smith@example.com

	course_id	course_name	credits	teacher_id
1	2	Mathematics	4	2
2	3	Physics	4	3
3	4	Chemistry	3	4
4	5	Computer Science	5	5
5	6	Biology	4	6
6	7	History	2	7
7	8	Geography	2	8
8	9	Political Science	3	9
9	10	Economics	4	10
10	12	Advanced Datab...	4	19

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.

```
import sys
import os

base_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), ".."))
sys.path.append(base_dir)

from util.DatabaseManager import DatabaseManager
from decimal import Decimal

db_manager = DatabaseManager()



first_name = 'Jane'
last_name = 'Johnson'

try:
    query = "SELECT student_id, outstanding_balance FROM Students WHERE first_name = ? AND
last_name = ?"
    student = db_manager.execute_query(query, (first_name, last_name))

    if student:
        student_id = student[0][0]
        outstanding_balance = student[0][1] if student[0][1] is not None else
Decimal('0.00')
        print(f"Found student ID {student_id} with outstanding balance
{outstanding_balance}")

        payment_amount = Decimal('500.00')
        payment_date = '2023-04-10'


        if outstanding_balance > Decimal('0.00'):

            db_manager.record_payment(student_id, payment_amount, payment_date)
            print(f"Recorded payment of {payment_amount} for student ID {student_id} on
{payment_date}")


            new_balance = outstanding_balance - payment_amount
            if new_balance < Decimal('0.00'):
                new_balance = Decimal('0.00')

            update_balance_query = "UPDATE Students SET outstanding_balance = ? WHERE
student_id = ?"
            db_manager.execute_query(update_balance_query, (new_balance, student_id))
            print(f"Updated outstanding balance to {new_balance} for student ID
{student_id}")
        else:
```

```
            print("Payment not required, outstanding balance is already 0 or negative.")
    else:
        print("Student not found.")
except Exception as e:
    print("Error processing payment:", e)
finally:
    db_manager.close()
```

Output:

```
Information System> & C:/Users/pumak/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/pumak/OneDrive/Desktop/Student Information System/m
ain/task10main.py"
Connected Successfully
Found student ID 12 with outstanding balance 2200.00
Inserted payment of 500.00 from student ID 12
Recorded payment of 500.00 for student ID 12 on 2023-04-10
Updated outstanding balance to 1700.00 for student ID 12
PS C:\Users\pumak\OneDrive\Desktop\Student Information System>
```

Updated student and payment records:

100 %

Results | Messages

	payment_id	student_id	amount	payment_date
1	15	12	500.00	2023-04-10

	student_id	first_name	last_name	date_of_birth	email	phone_number	outstanding_balance
1	12	Jane	Johnson	2001-06-15	jane.johnson@example.com	9876504321	1700.00

Task 11: Enrollment Report Generation

In this task, an administrator requests an enrollment report for a specific course, "Computer Science ." The system needs to retrieve enrollment information from the database and generate a report.

```
def generate_enrollment_report_for_course(self, course_id):
        course = self.database_manager.get_course_by_id(course_id)

        if not course:
            print(f"Course with ID '{course_id}' not found.")
            return

        enrollments = self.database_manager.get_enrollments_by_course(course_id)
        if not enrollments:
```

```
            print(f"No enrollments found for course ID '{course_id}'.")
            return

        print(f"\nEnrollment Report for Course ID '{course_id}':")
        print("--------------------------------------------------")
        for enrollment in enrollments:
            student_id, first_name, last_name, enrollment_date = enrollment
            print(f"Student ID: {student_id}, Name: {first_name} {last_name}, Enrollment
Date: {enrollment_date}")
        print("--------------------------------------------------")
```

```
import sys
import os

base_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), ".."))
sys.path.append(base_dir)

from entity.sis import SIS

if __name__ == "__main__":
    sis = SIS()
    course_id = 5
    sis.generate_enrollment_report_for_course(course_id)
```

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS                                   Python

PS C:\Users\pumak\OneDrive\Desktop\Student Information System> & C:/Users/pumak/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/pumak/OneDrive/Desktop/St
udent Information System/main/task11main.py"
Connected Successfully

Enrollment Report for Course ID '5':
--------------------------------------------------
Student ID: 3, Name: Karthik Venugopal, Enrollment Date: 2024-03-10
--------------------------------------------------
PS C:\Users\pumak\OneDrive\Desktop\Student Information System>
```