

Hexaware Foundation Training

Python Assignment

STUDENT INFORMATION SYSTEM

NAME: Aathirainathan P

DATE: 05-10-2024

Task 1: Define Classes

1.1 entity/student:

```
2 class Student:
3     def __init__(self, student_id, first_name, last_name, date_of_birth, email,
4         phone_number):
5         self.student_id = student_id
6         self.first_name = first_name
7         self.last_name = last_name
8         self.date_of_birth = date_of_birth
9         self.email = email
10        self.phone_number = phone_number
11
```

1.2 entity/course

```
11 class Course:
12     def __init__(self, course_id, course_name, course_code, instructor_name):
13         self.course_id = course_id
14         self.course_name = course_name
15         self.course_code = course_code
16         self.instructor_name = instructor_name
17
```

1.3 entity/enrollment

```
2 class Enrollment:
3     def __init__(self, enrollment_id, student_id, course_id, enrollment_date):
4         self.enrollment_id = enrollment_id
5         self.student_id = student_id
6         self.course_id = course_id
7         self.enrollment_date = enrollment_date
8
```

1.4 entity/teacher

```
2 class Teacher:
3     def __init__(self, teacher_id, first_name, last_name, email):
4         self.teacher_id = teacher_id
5         self.first_name = first_name
6         self.last_name = last_name
7         self.email = email
8
```

1.5 entity/payment

```
2 class Payment:
3     def __init__(self, payment_id, student_id, amount, payment_date):
4         self.payment_id = payment_id
5         self.student_id = student_id
6         self.amount = amount
7         self.payment_date = payment_date
8
```

1.6 SIS Class

```
2 class SIS:
3     def __init__(self):
4         self.students = []      # List to hold all students
5         self.courses = []      # List to hold all courses
6         self.teachers = []     # List to hold all teachers
7         self.enrollments = []  # List to hold all enrollments
8         self.payments = []     # List to hold all payments
```

Task 2: Implement Constructors

Already implemented in Task 1.

2.1 entity/student

```
1 class Student:
2     def __init__(self, student_id, first_name, last_name, date_of_birth, email,
3         phone_number):
4         self.student_id = student_id
5         self.first_name = first_name
6         self.last_name = last_name
7         self.date_of_birth = date_of_birth
8         self.email = email
9         self.phone_number = phone_number
```

2.2 entity/course

```
3 class Course:
4     def __init__(self, course_id, course_name, course_code, instructor_name):
5         self.course_id = course_id
```

```
6         self.course_name = course_name
7         self.course_code = course_code
8         self.instructor_name = instructor_name
9
```

2.3 entity/enrollment

```
9 class Enrollment:
10     def __init__(self, enrollment_id, student_id, course_id, enrollment_date):
11         self.enrollment_id = enrollment_id
12         self.student_id = student_id
13         self.course_id = course_id
14         self.enrollment_date = enrollment_date
15
```

2.4 entity/teacher

```
3 class Teacher:
4     def __init__(self, teacher_id, first_name, last_name, email):
5         self.teacher_id = teacher_id
6         self.first_name = first_name
7         self.last_name = last_name
8         self.email = email
9
```

2.5 entity/payment

```
9 class Payment:
10     def __init__(self, payment_id, student_id, amount, payment_date):
11         self.payment_id = payment_id
12         self.student_id = student_id
13         self.amount = amount
14         self.payment_date = payment_date
15
```

2.6 entity/ SIS

```
16 class SIS:
17     def __init__(self):
18         self.students = []          # List to hold all students
19         self.courses = []          # List to hold all courses
20         self.teachers = []         # List to hold all teachers
21         self.enrollments = []      # List to hold all enrollments
22         self.payments = []         # List to hold all payments
```

Task 3: Implement Methods

3.1 Student Class

```
23 import sys
24 import os
25
26 base_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), ".."))
27 sys.path.append(base_dir)
28
29 from entity.payment import Payment
30
31 class Student:
32     def __init__(self, student_id, first_name, last_name, date_of_birth, email,
33         phone_number):
34         self.student_id = student_id
35         self.first_name = first_name
36         self.last_name = last_name
37         self.date_of_birth = date_of_birth
38         self.email = email
39         self.phone_number = phone_number
40         self.enrolled_courses = [] # List to store enrolled courses
41         self.payments = [] # List to store payment records
42
43     def enroll_in_course(self, course):
44         self.enrolled_courses.append(course)
45
46     def update_student_info(self, first_name, last_name, date_of_birth, email,
47         phone_number):
48         self.first_name = first_name
49         self.last_name = last_name
50         self.date_of_birth = date_of_birth
51         self.email = email
52         self.phone_number = phone_number
53
54     def make_payment(self, amount, payment_date):
55         payment = Payment(None, self.student_id, amount, payment_date)
56         self.payments.append(payment)
57
58     def display_student_info(self):
59         print(f"Student ID: {self.student_id}")
60         print(f"Name: {self.first_name} {self.last_name}")
61         print(f>Date of Birth: {self.date_of_birth}")
62         print(f>Email: {self.email}")
63         print(f>Phone Number: {self.phone_number}")
64
65     def get_enrolled_courses(self):
66         return self.enrolled_courses
67
68     def get_payment_history(self):
69         return self.payments
```

3.2 Course Class:

```
68 class Course:
69     def __init__(self, course_id, course_name, course_code, instructor_name):
70         self.course_id = course_id
71         self.course_name = course_name
72         self.course_code = course_code
73         self.instructor_name = instructor_name
74         self.enrollments = [] # List to store enrollments
75         self.teacher = None
76
77     def assign_teacher(self, teacher):
78         self.teacher = teacher
79
80     def update_course_info(self, course_code, course_name, instructor):
81         self.course_code = course_code
82         self.course_name = course_name
83         self.instructor_name = instructor
84
85     def display_course_info(self):
86         print(f"Course ID: {self.course_id}")
87         print(f"Course Name: {self.course_name}")
88         print(f"Course Code: {self.course_code}")
89         print(f"Instructor Name: {self.instructor_name}")
90
91     def get_enrollments(self):
92         return self.enrollments
93
94     def get_teacher(self):
95         return self.teacher
```

3.3 Enrollment class:

```
96 class Enrollment:
97     def __init__(self, enrollment_id, student_id, course_id, enrollment_date):
98         self.enrollment_id = enrollment_id
99         self.student_id = student_id
100         self.course_id = course_id
101         self.enrollment_date = enrollment_date
102
103     def get_student_id(self):
104         return self.student_id
105
106     def get_course_id(self):
107         return self.course_id
```

3.4 Teacher Class:

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

```

112     self.last_name = last_name
113     self.email = email
114     self.assigned_courses = [] # List to store assigned courses
115
116     def update_teacher_info(self, first_name, last_name, email):
117         self.first_name = first_name
118         self.last_name = last_name
119         self.email = email
120
121     def display_teacher_info(self):
122         print(f"Teacher ID: {self.teacher_id}")
123         print(f"Name: {self.first_name} {self.last_name}")
124         print(f"Email: {self.email}")
125
126     def get_assigned_courses(self):
127         return self.assigned_courses

```

3.5 Payment Class:

```

128class Payment:
129    def __init__(self, payment_id, student_id, amount, payment_date):
130        self.payment_id = payment_id
131        self.student_id = student_id
132        self.amount = amount
133        self.payment_date = payment_date
134
135    def get_student(self):
136        return self.student_id
137
138    def get_payment_amount(self):
139        return self.amount
140
141    def get_payment_date(self):
142        return self.payment_date
143

```

3.6 SIS Class:

```

144import sys
145import os
146
147base_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), ".."))
148sys.path.append(base_dir)
149
150from entity.enrollment import Enrollment
151from entity.payment import Payment
152from entity.student import Student
153
154class SIS:
155    def __init__(self):
156        self.students = []
157        self.courses = []

```

```
158     self.enrollments = []
159     self.payments = []
160
161     def enroll_student_in_course(self, student, course):
162         enrollment_date = '2024-01-01'
163         enrollment = Enrollment(len(self.enrollments) + 1, student.student_id,
164                                 course.course_id, enrollment_date)
165         self.enrollments.append(enrollment)
166         print(f'Enrolled {student.first_name} in {course.course_name} on
167               {enrollment_date}')
168
169     def record_payment(self, student, amount, payment_date):
170         payment = Payment(len(self.payments) + 1, student.student_id, amount,
171                             payment_date)
172         self.payments.append(payment)
173         print(f'Recorded payment of {amount} from {student.first_name} on
174               {payment_date}')
175
176     def generate_enrollment_report(self, course):
177         enrollments = [enrollment for enrollment in self.enrollments if
178                         enrollment.course_id == course.course_id]
179         if not enrollments:
180             print(f"\nNo enrollments found for {course.course_name}")
181             return enrollments
182
183         print(f'\nEnrollment Report for {course.course_name}:')
184         for enrollment in enrollments:
185             print(f'Student ID: {enrollment.student_id}, Course ID:
186                   {enrollment.course_id}, Enrollment Date: {enrollment.enrollment_date}')
187
188         return enrollments
189
190     def generate_payment_report(self, student):
191         payment_report = [payment for payment in self.payments if payment.student_id ==
192                           student.student_id]
193         print(f'\nPayment Report for {student.first_name} {student.last_name}:')
194         for payment in payment_report:
195             print(f'Amount: {payment.amount}, Payment Date: {payment.payment_date}')
196
197     def calculate_course_statistics(self, course):
198         enrollments = [enrollment for enrollment in self.enrollments if
199                         enrollment.course_id == course.course_id]
200         if not enrollments:
201             print(f"\nNo enrollments found for {course.course_name}")
202             return 0, 0
203         total_payments = sum(payment.amount for payment in self.payments if
204                               payment.student_id in [e.student_id for e in enrollments])
205         return len(enrollments), total_payments
```

3.7 Testing the implemented methods:

```
197import sys
198import os
199
200base_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), ".."))
201sys.path.append(base_dir)
202
203from entity.student import Student
204from entity.course import Course
205from entity.teacher import Teacher
206from entity.enrollment import Enrollment
207from entity.payment import Payment
208from entity.sis import SIS
209
210def main():
211    sis = SIS()
212
213    student1 = Student(1, 'John', 'Doe', '1995-08-15', 'john.doe@example.com',
214        '1234567890')
215    student2 = Student(2, 'Jane', 'Smith', '1996-09-25', 'jane.smith@example.com',
216        '0987654321')
217
218    course1 = Course(1, 'Mathematics', 'MATH101', 'Dr. Alice')
219    course2 = Course(2, 'Physics', 'PHYS101', 'Dr. Bob')
220
221    sis.students.append(student1)
222    sis.students.append(student2)
223    sis.courses.append(course1)
224    sis.courses.append(course2)
225
226    sis.enroll_student_in_course(student1, course1)
227    sis.enroll_student_in_course(student2, course2)
228
229    print()
230
231    sis.record_payment(student1, 5000.00, '2024-01-20')
232    sis.record_payment(student2, 6000.00, '2024-02-15')
233
234    print()
235
236    sis.generate_enrollment_report(course2)
237    print()
238
239    sis.generate_payment_report(student1)
240
241    num_enrollments, total_payments = sis.calculate_course_statistics(course1)
242    print(f'\nStatistics for {course1.course_name}:')
243    print(f'Number of Enrollments: {num_enrollments}, Total Payments:
244        {total_payments}')
```


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/Student 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.

```
244import sys
245import os
246
247base_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), ".."))
248sys.path.append(base_dir)
249
250class DuplicateEnrollmentException(Exception):
251    def __init__(self, message="Student is already enrolled in this course."):
252        self.message = message
253        super().__init__(self.message)
254
255class CourseNotFoundException(Exception):
256    def __init__(self, message="Course not found in the system."):
257        self.message = message
258        super().__init__(self.message)
259
260class StudentNotFoundException(Exception):
261    def __init__(self, message="Student not found in the system."):
262        self.message = message
263        super().__init__(self.message)
264
265class TeacherNotFoundException(Exception):
266    def __init__(self, message="Teacher not found in the system."):
267        self.message = message
268        super().__init__(self.message)
269
270class PaymentValidationException(Exception):
271    def __init__(self, message="Payment validation failed."):
272        self.message = message
273        super().__init__(self.message)
```

```

274
275class InvalidStudentDataException(Exception):
276    def __init__(self, message="Invalid data provided for the student."):
277        self.message = message
278        super().__init__(self.message)
279
280class InvalidCourseDataException(Exception):
281    def __init__(self, message="Invalid data provided for the course."):
282        self.message = message
283        super().__init__(self.message)
284
285class InvalidEnrollmentDataException(Exception):
286    def __init__(self, message="Invalid data provided for the enrollment."):
287        self.message = message
288        super().__init__(self.message)
289
290class InvalidTeacherDataException(Exception):
291    def __init__(self, message="Invalid data provided for the teacher."):
292        self.message = message
293        super().__init__(self.message)
294
295class InsufficientFundsException(Exception):
296    def __init__(self, message="Insufficient funds for enrollment."):
297        self.message = message
298        super().__init__(self.message)
299

```

Task 5: Collections

Implement Collections:

5.1 Student Class:

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

```

300class Student:
301    def __init__(self, student_id, first_name, last_name, date_of_birth, email,
302                phone_number):
303        self.student_id = student_id
304        self.first_name = first_name
305        self.last_name = last_name
306        self.date_of_birth = date_of_birth
307        self.email = email
308        self.phone_number = phone_number
309        self.enrolled_courses = [] # List to store enrolled courses
310        self.payments = [] # List to store payment records
311
312    def enroll_in_course(self, course):
313        self.enrolled_courses.append(course)
314
315    def get_enrolled_courses(self):
316        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`.

```
316 class Course:
317     def __init__(self, course_id, course_name, course_code, instructor_name):
318         self.course_id = course_id
319         self.course_name = course_name
320         self.course_code = course_code
321         self.instructor_name = instructor_name
322         self.enrollments = [] # List to store Enrollment objects
323         self.teacher = None
324
325     def enroll_student(self, student, enrollment_date):
326         enrollment = Enrollment(len(self.enrollments) + 1, student.student_id,
327         self.course_id, enrollment_date)
328         self.enrollments.append(enrollment)
329
330     def get_enrollments(self):
331         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.

```
332class Teacher:
333    def __init__(self, teacher_id, first_name, last_name, email):
334        self.teacher_id = teacher_id
335        self.first_name = first_name
336        self.last_name = last_name
337        self.email = email
338        self.assigned_courses = [] # List to store assigned courses
339
340    def assign_course(self, course):
341        if course not in self.assigned_courses:
342            self.assigned_courses.append(course)
343            course.assign_teacher(self) # Assuming Course has an assign_teacher method
344        else:
345            print(f"{self.first_name} is already assigned to {course.course_name}.")
346
347    def get_assigned_courses(self):
348        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:

```

349 def AddEnrollment(self, student, course, enrollment_date):
350
351     if any(enrollment.student_id == student.student_id and enrollment.course_id ==
        course.course_id for enrollment in self.enrollments):
352         raise Exception("Student is already enrolled in this course.")
353
354     enrollment = Enrollment(len(self.enrollments) + 1, student.student_id,
        course.course_id, enrollment_date)
355
356     enrollment.set_student(student)
357     enrollment.set_course(course)
358
359     self.enrollments.append(enrollment)
360
361     student.enrolled_courses.append(course)
362     course.enrollments.append(enrollment)
363
364     print(f'Enrolled {student.first_name} in {course.course_name} on
        {enrollment_date}')

```

6.2 AssignCourseToTeacher:

```

365 def assign_course_to_teacher(self, course, teacher):
366     if course not in teacher.assigned_courses:
367         teacher.assigned_courses.append(course)
368         print(f'Course {course.course_name} has been assigned to teacher
            {teacher.first_name} {teacher.last_name}.')
369     else:
370         print(f'Course {course.course_name} is already assigned to teacher
            {teacher.first_name} {teacher.last_name}.')
371

```

6.3 AddPayment:

```

372 def add_payment(self, student, amount, payment_date):
373
374     payment = Payment(len(self.payments) + 1, student.student_id, amount,
        payment_date)

```

```
375
376     student.make_payment(amount, payment_date)
377
378     self.payments.append(payment)
379
380     print(f'Recorded payment of {amount} from {student.first_name} on
    {payment_date}')
```

6.4 GetEnrollmentsForStudent:

```
381def GetEnrollmentsForStudent(self, student):
382     return [enrollment for enrollment in self.enrollments if
    enrollment.student.student_id == student.student_id]
383
```

6.5 GetCoursesForTeacher:

```
384     def GetCoursesForTeacher(self, teacher):
385         return [course for course in self.courses if course in
    teacher.assigned_courses]
386
```

6.6 Main Method:

```
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():
```

```
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
PS C:\Users\pumak\OneDrive\Desktop\Student Information System> & C:/Users/pumak/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/pumak/OneDrive/Desktop/Student 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:

```
387class DBPropertyUtil:
388    @staticmethod
389    def get_connection_string():
390        return 'Driver={SQL
        Server};Server=PUMA\\SQLEXPRESS;Database=SISDB;Trusted_Connection=yes; '
391
392
```

7.1.2 util/db_conn_util:

```
393# DBConnUtil.py
394import sys
395import os
396
397base_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), ".."))
398sys.path.append(base_dir)
399
400import pyodbc
401from util.db_property_util import DBPropertyUtil
402
403class DBConnUtil:
404    @staticmethod
405    def get_connection():
406        connection_string = DBPropertyUtil.get_connection_string()
407        try:
408            conn = pyodbc.connect(connection_string)
409            print("Connected Successfully")
```

```

410         return conn
411     except Exception as e:
412         print("Connection failed:", e)
413         return None
414
415

```

7.1. util/DatabaseManager(For DB Initialization):

```

416import sys
417import os
418
419base_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), ".."))
420sys.path.append(base_dir)
421import pyodbc
422from util.db_conn_util import DBConnUtil
423
424class DatabaseManager:
425    def __init__(self):
426        self.conn = DBConnUtil.get_connection()
427        if self.conn:
428            self.cursor = self.conn.cursor()
429            self.initialize_database()
430
431    def initialize_database(self):
432        sql_commands = [
433            '''
434            IF NOT EXISTS (SELECT * FROM sysobjects WHERE name='Students' AND
435            xtype='U')
436            CREATE TABLE Students (
437                id INT PRIMARY KEY IDENTITY(1,1),
438                first_name NVARCHAR(100) NOT NULL,
439                last_name NVARCHAR(100) NOT NULL,
440                dob DATE NOT NULL,
441                email NVARCHAR(255) NOT NULL UNIQUE,
442                phone NVARCHAR(15) NOT NULL
443            );
444            ''',
445            IF NOT EXISTS (SELECT * FROM sysobjects WHERE name='Courses' AND xtype='U')
446            CREATE TABLE Courses (
447                id INT PRIMARY KEY IDENTITY(1,1),
448                course_name NVARCHAR(100) NOT NULL,
449                course_code NVARCHAR(20) NOT NULL UNIQUE,
450                instructor_name NVARCHAR(100) NOT NULL
451            );
452            ''',
453            IF NOT EXISTS (SELECT * FROM sysobjects WHERE name='Enrollments' AND
454            xtype='U')
455            CREATE TABLE Enrollments (
456                id INT PRIMARY KEY IDENTITY(1,1),
457                student_id INT NOT NULL,

```

```

458         course_id INT NOT NULL,
459         enrollment_date DATE NOT NULL,
460         FOREIGN KEY (student_id) REFERENCES Students (id),
461         FOREIGN KEY (course_id) REFERENCES Courses (id),
462         UNIQUE (student_id, course_id)
463     );
464     '''
465     '''
466     IF NOT EXISTS (SELECT * FROM sysobjects WHERE name='Teachers' AND
xtype='U')
467     CREATE TABLE Teachers (
468         id INT PRIMARY KEY IDENTITY(1,1),
469         first_name NVARCHAR(100) NOT NULL,
470         last_name NVARCHAR(100) NOT NULL,
471         email NVARCHAR(255) NOT NULL UNIQUE
472     );
473     '''
474     '''
475     IF NOT EXISTS (SELECT * FROM sysobjects WHERE name='Payments' AND
xtype='U')
476     CREATE TABLE Payments (
477         id INT PRIMARY KEY IDENTITY(1,1),
478         student_id INT NOT NULL,
479         amount DECIMAL(10, 2) NOT NULL,
480         payment_date DATE NOT NULL,
481         FOREIGN KEY (student_id) REFERENCES Students (id)
482     );
483     '''
484 ]
485
486 for command in sql_commands:
487     self.cursor.execute(command)
488
489     self.conn.commit()
490
491 def close(self):
492     if self.conn:
493         self.conn.close()
494
495

```

7.2 Data Retrieval:

```

496 def dynamic_query(self, table, columns=None, conditions=None, order_by=None):
497     try:
498         columns = ', '.join(columns) if columns else '*'
499         query = f"SELECT {columns} FROM {table}"
500         if conditions:
501             query += " WHERE " + ' AND '.join(conditions)
502         if order_by:
503             query += " ORDER BY " + order_by
504
505         self.cursor.execute(query)

```

```

506         return self.cursor.fetchall()
507     except Exception as e:
508         print("Error executing dynamic query:", e)
509         return []
510
511     def get_students(self):
512         return self.dynamic_query("Students")
513
514     def get_courses(self):
515         return self.dynamic_query("Courses")
516
517     def get_enrollments(self):
518         return self.dynamic_query("Enrollments")
519
520     def get_teachers(self):
521         return self.dynamic_query("Teachers")
522
523     def get_payments(self):
524         return self.dynamic_query("Payments")
525

```

7.3 Data Insertion and Updating:

```

526 def insert_student(self, first_name, last_name, dob, email, phone):
527     try:
528         self.cursor.execute(
529             "INSERT INTO Students (first_name, last_name, dob, email, phone) VALUES
530             (?, ?, ?, ?, ?)",
531             (first_name, last_name, dob, email, phone)
532         )
533         self.conn.commit()
534         print(f"Inserted student: {first_name} {last_name}")
535     except Exception as e:
536         print("Error inserting student:", e)
537
538     def update_student(self, student_id, first_name, last_name, dob, email, phone):
539         try:
540             self.cursor.execute(
541                 "UPDATE Students SET first_name = ?, last_name = ?, dob = ?, email = ?,
542                 phone = ? WHERE id = ?",
543                 (first_name, last_name, dob, email, phone, student_id)
544             )
545             self.conn.commit()
546             print(f"Updated student ID {student_id}")
547         except Exception as e:
548             print("Error updating student:", e)
549
550     def insert_enrollment(self, student_id, course_id, enrollment_date):
551         try:
552             self.cursor.execute(
553                 "INSERT INTO Enrollments (student_id, course_id, enrollment_date)
554                 VALUES (?, ?, ?)",
555                 (student_id, course_id, enrollment_date)
556             )
557             self.conn.commit()
558             print(f"Inserted enrollment for student {student_id} on {enrollment_date}")
559         except Exception as e:
560             print("Error inserting enrollment:", e)
561

```

```

553         )
554         self.conn.commit()
555         print(f"Inserted enrollment for student ID {student_id} in course ID
{course_id}")
556     except Exception as e:
557         print("Error inserting enrollment:", e)
558
559     def record_payment(self, student_id, amount, payment_date):
560         try:
561             self.cursor.execute(
562                 "INSERT INTO Payments (student_id, amount, payment_date) VALUES (?, ?,
?)"',
563                 (student_id, amount, payment_date)
564             )
565             self.conn.commit()
566             print(f"Inserted payment of {amount} from student ID {student_id}")
567         except Exception as e:
568             print("Error inserting payment:", e)
569

```

7.4 Transaction Management:

```

570 def begin_transaction(self):
571     self.conn.autocommit = False
572
573     def commit_transaction(self):
574         self.conn.commit()
575         self.conn.autocommit = True
576
577     def rollback_transaction(self):
578         self.conn.rollback()
579         self.conn.autocommit = True
580
581 def insert_student(self, first_name, last_name, dob, email, phone):
582     try:
583         self.cursor.execute(
584             "INSERT INTO Students (first_name, last_name, dob, email, phone) VALUES
(?, ?, ?, ?, ?)"',
585             (first_name, last_name, dob, email, phone)
586         )
587         self.conn.commit()
588         print(f"Inserted student: {first_name} {last_name}")
589     except Exception as e:
590         print("Error inserting student:", e)
591
592     def update_student(self, student_id, first_name, last_name, dob, email, phone):
593         try:
594             self.cursor.execute(
595                 "UPDATE Students SET first_name = ?, last_name = ?, dob = ?, email = ?,
phone = ? WHERE id = ?" ,
596                 (first_name, last_name, dob, email, phone, student_id)
597             )
598             self.conn.commit()

```

```

599         print(f"Updated student ID {student_id}")
600     except Exception as e:
601         print("Error updating student:", e)
602
603     def insert_enrollment(self, student_id, course_id, enrollment_date):
604         try:
605             self.cursor.execute(
606                 "INSERT INTO Enrollments (student_id, course_id, enrollment_date)
VALUES (?, ?, ?)",
607                 (student_id, course_id, enrollment_date)
608             )
609             self.conn.commit()
610             print(f"Inserted enrollment for student ID {student_id} in course ID
{course_id}")
611         except Exception as e:
612             print("Error inserting enrollment:", e)
613
614     def record_payment(self, student_id, amount, payment_date):
615         try:
616             self.cursor.execute(
617                 "INSERT INTO Payments (student_id, amount, payment_date) VALUES (?, ?,
?)",
618                 (student_id, amount, payment_date)
619             )
620             self.conn.commit()
621             print(f"Inserted payment of {amount} from student ID {student_id}")
622         except Exception as e:
623             print("Error inserting payment:", e)
624
625     def enroll_student_with_payment(self, student_id, course_id, enrollment_date,
amount, payment_date):
626         self.begin_transaction()
627         try:
628             self.insert_enrollment(student_id, course_id, enrollment_date)
629             self.record_payment(student_id, amount, payment_date)
630             self.commit_transaction()
631         except Exception as e:
632             self.rollback_transaction()
633             print("Transaction failed:", e)
634
635

```

7.5 Dynamic Query Builder:

```

636 def dynamic_query(self, table, columns=None, conditions=None, order_by=None):
637     try:
638         columns = ', '.join(columns) if columns else '*'
639         query = f"SELECT {columns} FROM {table}"
640         if conditions:
641             query += " WHERE " + ' AND '.join(conditions)
642         if order_by:
643             query += " ORDER BY " + order_by
644

```

```
645         self.cursor.execute(query)
646         return self.cursor.fetchall()
647     except Exception as e:
648         print("Error executing dynamic query:", e)
649         return []
650
```

Task 8: Student Enrollment:

```
651import sys
652import os
653
654base_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), ".."))
655sys.path.append(base_dir)
656
657from util.DatabaseManager import DatabaseManager
658from util.db_conn_util import DBConnUtil
659from entity.student import Student
660from entity.course import Course
661from entity.enrollment import Enrollment
662from entity.sis import SIS
663from entity.teacher import Teacher
664from entity.payment import Payment
665from exception.custom_exceptions import (
666    DuplicateEnrollmentException,
667    CourseNotFoundException,
668    StudentNotFoundException,
669    TeacherNotFoundException,
670    PaymentValidationException,
671    InvalidStudentDataException,
672    InvalidCourseDataException,
673    InvalidEnrollmentDataException,
674    InvalidTeacherDataException,
675    InsufficientFundsException
676)
677
678if __name__ == "__main__":
679    db_manager = DatabaseManager()
680
681    first_name = 'John'
682    last_name = 'Doe'
683    dob = '1995-08-15'
684    email = 'john.doe@example.com'
685    phone = '123-456-7890'
686
687    db_manager.insert_student(first_name, last_name, dob, email, phone)
688
689    course_names = ['Introduction to Programming', 'Mathematics 101']
690    course_ids = []
691
692    for course_name in course_names:
693
```

```

694     course = db_manager.dynamic_query(
695         "Courses",
696         columns=["course_id"],
697         conditions=[f"course_name = '{course_name}'"]
698     )
699     if course:
700         course_ids.append(course[0][0])
701
702     student_id = db_manager.dynamic_query("Students", columns=["student_id"],
703     conditions=[f"email = '{email}'"])[0][0]
704
705     enrollment_date = "2024-01-01"
706     for course_id in course_ids:
707         db_manager.insert_enrollment(student_id, course_id, enrollment_date)
708
709     print(f"John Doe has been enrolled in the following courses: {course_names}")
710
711     db_manager.close()
712

```

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)

```



```

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/pumak/OneDrive/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/main/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

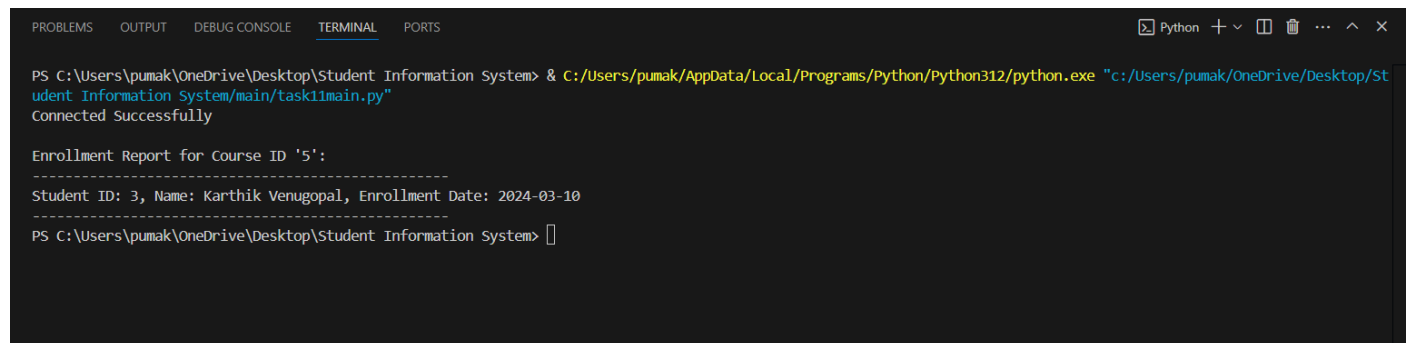
base_dir = os.path.abspath(os.path.join(os.path.dirname(__file__), ".."))
sys.path.append(base_dir)

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 + - [ ] [ ] ... ^ x

PS C:\Users\pumak\OneDrive\Desktop\Student Information System> & C:/Users/pumak/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/pumak/OneDrive/Desktop/Student 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> 

```