STUDENT INFORMATION SYSTEM – OOPS IMPLEMENTATION

TASK 1 & 2 – Defining class and Implementing constructor

1. 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
        self.enrollments = []
        self.payments = []
```

Student.py

entity > Student.py > Student > __init__

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

2. Course :

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

Student.py Course.py

entity > Course.py > Course > __init__

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

3. 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
        self.assigned_courses = []

```
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 = []
```

4. Enrollment:

class Enrollment:
    def __init__(self, enrollment_id, student, course, enrollment_date):
        self.enrollment_id = enrollment_id
        self.student = student
        self.course = course
        self.enrollment_date = enrollment_date

```
entity > Enrollment.py > Enrollment > __init__
class Enrollment:
    def __init__(self, enrollment_id, student, course, enrollment_date):
        self.enrollment_id = enrollment_id
        self.student = student
        self.course = course
        self.enrollment_date = enrollment_date
```

5. Payment :

class Payment:
    def __init__(self, payment_id, student, amount, payment_date):
        self.payment_id = payment_id
        self.student = student # Student object
        self.amount = amount
        self.payment_date = payment_date

```
class Payment:
    def __init__(self, payment_id, student, amount, payment_date):
        self.payment_id = payment_id
        self.student = student
        self.amount = amount
        self.payment_date = payment_date
```

TASK 3 – Implementing Methods

1. Student :

```
def enroll_in_course(self, course, enrollment_date):
    from entity.Enrollment import Enrollment
    enrollment = Enrollment(None, self, course, enrollment_date)
    self.enrollments.append(enrollment)
    return enrollment

def update_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):
    from entity.Payment import Payment
    payment = Payment(None, self, amount, payment_date)
    self.payments.append(payment)
    return payment

def display_info(self):
    print(f"Student ID: {self.student_id}")
    print(f"Name: {self.first_name} {self.last_name}")
    print(f"DOB: {self.date_of_birth}")
    print(f"Email: {self.email}")
    print(f"Phone: {self.phone_number}")

def get_enrolled_courses(self):
    return [enrollment.course for enrollment in self.enrollments]

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

Student.py × Course.py

entity > Student.py > Student > __init__

```python
class Student:

    def enroll_in_course(self, course, enrollment_date):
        from entity.Enrollment import Enrollment
        enrollment = Enrollment(None, self, course, enrollment_date)
        self.enrollments.append(enrollment)
        return enrollment

    def update_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):
        from entity.Payment import Payment
        payment = Payment(None, self, amount, payment_date)
        self.payments.append(payment)
        return payment

    def display_info(self):
        print(f"Student ID: {self.student_id}")
        print(f"Name: {self.first_name} {self.last_name}")
        print(f"DOB: {self.date_of_birth}")
        print(f"Email: {self.email}")
        print(f"Phone: {self.phone_number}")

    def get_enrolled_courses(self):
        return [enrollment.course for enrollment in self.enrollments]

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

2. Course :

```python
def assign_teacher(self, teacher):
    self.teacher = teacher
    teacher.assigned_courses.append(self)

def update_info(self, course_code, course_name, teacher):
    self.course_code = course_code
    self.course_name = course_name
    self.teacher = teacher

def display_info(self):
    print(f"Course ID: {self.course_id}")
    print(f"Name: {self.course_name}")
    print(f"Code: {self.course_code}")
    if self.teacher:
        print(f"Instructor: {self.teacher.first_name} {self.teacher.last_name}")

def get_enrollments(self):
    return self.enrollments

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

```python
entity > Course.py > Course > __init__
class Course:
    def __init__(self, course_id, course_name, course_code, teacher=None):
        self.course_id = course_id
        self.course_name = course_name
        self.course_code = course_code
        self.teacher = teacher
        self.enrollments = []

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

    def update_info(self, course_code, course_name, teacher):
        self.course_code = course_code
        self.course_name = course_name
        self.teacher = teacher

    def display_info(self):
        print(f"Course ID: {self.course_id}")
        print(f"Name: {self.course_name}")
        print(f"Code: {self.course_code}")
        if self.teacher:
            print(f"Instructor: {self.teacher.first_name} {self.teacher.last_name}")

    def get_enrollments(self):
        return self.enrollments

    def get_teacher(self):
        return self.teacher

```

3. Teacher:

```python
def update_info(self, first_name, last_name, email):
    self.first_name = first_name
    self.last_name = last_name
    self.email = email
def display_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
```

```python
entity > Teacher.py > ...
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 = []

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

    def display_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

```

4. Enrollment:

```
def get_student(self):
    return self.student
def get_course(self):
    return self.course
```

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

    def get_student(self):
        return self.student

    def get_course(self):
        return self.course
```

5. Payment :

```
def get_student(self):
    return self.student
def get_payment_amount(self):
    return self.amount
def get_payment_date(self):
    return self.payment_date
```

entity > Payment.py > Payment > __init__

```python
class Payment:
    def __init__(self, payment_id, student, amount, payment_date):
        self.payment_id = payment_id
        self.student = student
        self.amount = amount
        self.payment_date = payment_date

    def get_student(self):
        return self.student

    def get_payment_amount(self):
        return self.amount

    def get_payment_date(self):
        return self.payment_date

```

Task 4 - Exceptions handling and Custom Exceptions

```python
class SISException(Exception):
    pass

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

class CourseNotFoundException(SISException):
    def __init__(self, message="Course not found."):
        super().__init__(message)

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

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

class PaymentValidationException(SISException):
    def __init__(self, message="Invalid payment details."):
        super().__init__(message)

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

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

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

```

```python

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

class InsufficientFundsException(SISException):
    def __init__(self, message="Insufficient funds to enroll in course."):
        super().__init__(message)

```

Task 5 – Collections

1. 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
        self.enrollments = []
        self.payments = []
```

Student.py ×

entity > Student.py > Student > __init__

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

```

2. Course :

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

```python
class Course:
    def __init__(self, course_id, course_name, course_code, teacher=None):
        self.course_id = course_id
        self.course_name = course_name
        self.course_code = course_code
        self.teacher = teacher
        self.enrollments = []
```

3. Teacher:

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

4. Enrollment:

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

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

5. Payment :

```
class Payment:
    def __init__(self, payment_id, student, amount, payment_date):
        self.payment_id = payment_id
        self.student = student  # Student object
        self.amount = amount
        self.payment_date = payment_date
```

```python
class Payment:
    def __init__(self, payment_id, student, amount, payment_date):
        self.payment_id = payment_id
        self.student = student
        self.amount = amount
        self.payment_date = payment_date
```

TASK 6

Main Driver Code :

```python
from dao.implementation import SISDAOImpl
from entity.student import Student
from entity.teacher import Teacher
from entity.course import Course
from exception.custom_exceptions import StudentAlreadyExistsException, CourseAlreadyExistsException, TeacherAlreadyExistsException,
import mysql.connector
def main():
    sis = SISDAOImpl()

    print("\n Add Student John Doe")
    john = Student(101, "John", "Doe", "1995-08-15", "john.doe@example.com", "1234567890")
    try:
        sis.add_student(john)
        print(" John Doe inserted successfully.")
    except StudentAlreadyExistsException:
        print("  John Doe already exists.")


    print("\n Add Courses")
    intro = Course(1, "Introduction to Programming", "CS101")
    math = Course(2, "Mathematics 101", "MTH101")
    try:
        sis.add_course(intro)
        print(" Course 'Introduction to Programming' inserted.")
    except CourseAlreadyExistsException:
        print("  Course 'Introduction to Programming' already exists.")
    try:
        sis.add_course(math)
        print(" Course 'Mathematics 101' inserted.")
    except CourseAlreadyExistsException:
        print("  Course 'Mathematics 101' already exists.")
```

```python
    print("\n Enroll John in 2 Courses")
    try:
        sis.enroll_student(101, 1, "2024-05-01")
        print(" John enrolled in 'Introduction to Programming'.")
    except DuplicateEnrollmentException:
        print("  John already enrolled in 'Introduction to Programming'.")
    try:
        sis.enroll_student(101, 2, "2024-05-01")
        print(" John enrolled in 'Mathematics 101'.")
    except DuplicateEnrollmentException:
        print("  John already enrolled in 'Mathematics 101'.")

    print("\n Assign Teacher Sarah Smith to Advanced DB Course")
    sarah = Teacher(201, "Sarah", "Smith", "sarah.smith@example.com")
    adv_db = Course(3, "Advanced Database Management", "CS302")
    try:
        sis.add_teacher(sarah)
        print(" Teacher Sarah Smith added.")
    except TeacherAlreadyExistsException:
        print("  Sarah Smith already exists.")
    try:
        sis.add_course(adv_db)
        print(" Course 'Advanced Database Management' added.")
    except CourseAlreadyExistsException:
        print("  Course 'Advanced Database Management' already exists.")
    try:
        sis.assign_teacher_to_course(3, 201)
        print(" Sarah Smith assigned to CS302.")
    except Exception as e:
        print(f"  Could not assign Sarah: {e}")
```

```python
    print("\n Record Payment for Jane Johnson")
    jane = Student(102, "Jane", "Johnson", "1997-06-10", "jane.j@example.com", "9876543210")
    try:
        sis.add_student(jane)
        print(" Jane Johnson added.")
    except StudentAlreadyExistsException:
        print("  Jane Johnson already exists.")
    try:
        sis.record_payment(102, 500.00, "2023-04-10")
        print(" Payment recorded for Jane.")
    except Exception as e:
        print(f" Could not record payment: {e}")

    print("\n Enrollment Report for 'Introduction to Programming'")
    enrolled = sis.get_enrollments_by_course("Introduction to Programming")
    if enrolled:
        for entry in enrolled:
            print(f"{entry[0]} {entry[1]} - Enrolled on: {entry[2]}")
    else:
        print(" No enrollments found for this course.")


    print("\n Enrollment Report for 'Computer Science 101'")
    enrolled = sis.get_enrollments_by_course("Computer science 101")
    if enrolled:
        for entry in enrolled:
            print(f"{entry[0]} {entry[1]} - Enrolled on: {entry[2]}")
    else:
        print("  No enrollments found for this course.")

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

DAO / SIS :

Dao/interface :

```python
from abc import ABC, abstractmethod
class SISDAO(ABC):
    @abstractmethod
    def add_student(self, student): pass

    @abstractmethod
    def add_course(self, course): pass

    @abstractmethod
    def add_teacher(self, teacher): pass

    @abstractmethod
    def enroll_student(self, student_id, course_id, date): pass

    @abstractmethod
    def record_payment(self, student_id, amount, payment_date): pass

    @abstractmethod
    def assign_teacher_to_course(self, course_id, teacher_id): pass

    @abstractmethod
    def get_enrollments_by_course(self, course_name): pass
```

Dao/implementation :

```python
import mysql.connector
from util.db_conn_util import DBConnUtil
from entity.student import Student
from entity.teacher import Teacher
from entity.course import Course
from exception.custom_exceptions import *

class SISDAOImpl:
    def __init__(self):
        #conn_str = DBPropertyUtil.get_connection_string()
        #self.conn = DBConnUtil.get_connection(conn_str)
        self.conn = DBConnUtil.get_connection()

        #self.conn.execute("PRAGMA foreign_keys = ON;")


    def add_student(self, student: Student):
        cursor = self.conn.cursor()
        try:
            cursor.execute("""
                INSERT INTO Students (student_id, first_name, last_name, date_of_birth, email, phone_number)
                VALUES (%s, %s, %s, %s, %s, %s)
            """, (
                student.student_id, student.first_name, student.last_name,
                student.date_of_birth, student.email, student.phone_number
            ))
            self.conn.commit()
        except mysql.connector.IntegrityError as e:
            if "Duplicate entry" in str(e):
                raise StudentAlreadyExistsException()
            else:
                raise
```

```python
    def add_course(self, course: Course):
        cursor = self.conn.cursor()
        try:
            cursor.execute("""
                INSERT INTO Courses (course_id, course_name, course_code)
                VALUES (%s, %s, %s)
            """, (
                course.course_id, course.course_name, course.course_code
            ))
            self.conn.commit()
        except mysql.connector.IntegrityError as e:
            if "Duplicate entry" in str(e):
                raise CourseAlreadyExistsException()
            else:
                raise

    def add_teacher(self, teacher: Teacher):
        cursor = self.conn.cursor()
        try:
            cursor.execute("""
                INSERT INTO Teacher (teacher_id, first_name, last_name, email)
                VALUES (%s, %s, %s, %s)
            """, (
                teacher.teacher_id, teacher.first_name, teacher.last_name, teacher.email
            ))
            self.conn.commit()
        except mysql.connector.IntegrityError as e:
            if "Duplicate entry" in str(e):
                raise TeacherAlreadyExistsException()
            else:
                raise
```

```python
def enroll_student(self, student_id, course_id, date):
    cursor = self.conn.cursor()
    cursor.execute("SELECT * FROM Enrollments WHERE student_id = %s AND course_id =%s", (student_id, course_id))
    if cursor.fetchone():
        raise DuplicateEnrollmentException()
    cursor.execute("INSERT INTO Enrollments (student_id, course_id, enrollment_date) VALUES (%s, %s, %s)",
                   (student_id, course_id, date))
    self.conn.commit()

def record_payment(self, student_id, amount, payment_date):
    if amount <= 0:
        raise PaymentValidationException()
    cursor = self.conn.cursor()
    cursor.execute("INSERT INTO Payments (student_id, amount, payment_date) VALUES (%s, %s, %s)",
                   (student_id, amount, payment_date))
    self.conn.commit()

def assign_teacher_to_course(self, course_id, teacher_id):
    cursor = self.conn.cursor()
    cursor.execute("UPDATE Courses SET teacher_id = %s WHERE course_id = %s",
                   (teacher_id, course_id))
    self.conn.commit()

def get_enrollments_by_course(self, course_name):
    cursor = self.conn.cursor()
    cursor.execute("""SELECT s.first_name, s.last_name, e.enrollment_date
                      FROM Enrollments e
                      JOIN Students s ON s.student_id = e.student_id
                      JOIN Courses c ON c.course_id = e.course_id
                      WHERE c.course_name = %s""", (course_name,))
    return cursor.fetchall()
```

Task 7 - Database Connectivity

```python
import mysql.connector
class DBConnUtil:
    @staticmethod
    def get_connection():
        return mysql.connector.connect(
            host="localhost",
            user="root",
            password="dharshini2004",
            database="SIS"
        )
```

db_conn_util.py > ...

```python
import mysql.connector

class DBConnUtil:
    @staticmethod
    def get_connection():
        return mysql.connector.connect(
            host="localhost",
            user="root",
            password="dharshini2004",
            database="SIS"
        )
```

Task 8 - Student Enrollment

John Doe's details:

• First Name: John
• Last Name: Doe
• Date of Birth: 1995-08-15
• Email: john.doe@example.com
• Phone Number: 123-456-7890

```python
def main():
    sis = SISDAOImpl()

    print("\n Add Student John Doe")
    john = Student(101, "John", "Doe", "1995-08-15", "john.doe@example.com", "1234567890")
    try:
        sis.add_student(john)
        print(" John Doe inserted successfully.")
    except StudentAlreadyExistsException:
        print("  John Doe already exists.")
```

John is enrolling in the following courses:

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

```python
    print("\n Enroll John in 2 Courses")
    try:
        sis.enroll_student(101, 1, "2024-05-01")
        print(" John enrolled in 'Introduction to Programming'.")
    except DuplicateEnrollmentException:
        print("  John already enrolled in 'Introduction to Programming'.")
    try:
        sis.enroll_student(101, 2, "2024-05-01")
        print(" John enrolled in 'Mathematics 101'.")
    except DuplicateEnrollmentException:
        print("  John already enrolled in 'Mathematics 101'.")
```

Task 9 - Teacher Assignment

Teacher's Details:

• Name: Sarah Smith
• Email: sarah.smith@example.com
• Expertise: Computer Science

```python
print("\n Assign Teacher Sarah Smith to Advanced DB Course")
sarah = Teacher(201, "Sarah", "Smith", "sarah.smith@example.com")
adv_db = Course(3, "Advanced Database Management", "CS302")
try:
    sis.add_teacher(sarah)
    print(" Teacher Sarah Smith added.")
except TeacherAlreadyExistsException:
    print("  Sarah Smith already exists.")
try:
    sis.add_course(adv_db)
    print(" Course 'Advanced Database Management' added.")
except CourseAlreadyExistsException:
    print("  Course 'Advanced Database Management' already exists.")
try:
    sis.assign_teacher_to_course(3, 201)
    print(" Sarah Smith assigned to CS302.")
except Exception as e:
    print(f"  Could not assign Sarah: {e}")
```

Course to be assigned:

• Course Name: Advanced Database Management

• Course Code: CS302

```python
print("\n Add Courses")
intro = Course(1, "Introduction to Programming", "CS101")
math = Course(2, "Mathematics 101", "MTH101")
try:
    sis.add_course(intro)
    print(" Course 'Introduction to Programming' inserted.")
except CourseAlreadyExistsException:
    print("  Course 'Introduction to Programming' already exists.")
try:
    sis.add_course(math)
    print(" Course 'Mathematics 101' inserted.")
except CourseAlreadyExistsException:
    print("  Course 'Mathematics 101' already exists.")
```

Task 10 - Payment Record

Jane Johnson's details:

• Student ID: 102

• Payment Amount: $500.00

• Payment Date: 2023-04-10

```python
print("\n 5. Record Payment for Jane Johnson")
jane = Student(102, "Jane", "Johnson", "1997-06-10", "jane.j@example.com", "9876543210")
try:
    sis.add_student(jane)
    print(" Jane Johnson added.")
except StudentAlreadyExistsException:
    print("  Jane Johnson already exists.")
try:
    sis.record_payment(102, 500.00, "2023-04-10")
    print(" Payment recorded for Jane.")
except Exception as e:
    print(f" Could not record payment: {e}")
```

Task 11 - Enrollment Report Generation

Course to generate the report for:

• Course Name: Computer Science 101

```
print("\n 6.Enrollment Report for 'Introduction to Programming'")
enrolled = sis.get_enrollments_by_course("Introduction to Programming")
if enrolled:
    for entry in enrolled:
        print(f"{entry[0]} {entry[1]} - Enrolled on: {entry[2]}")
else:
    print(" No enrollments found for this course.")


print("\n 7. Enrollment Report for 'Computer Science 101'")
enrolled = sis.get_enrollments_by_course("Computer science 101")
if enrolled:
    for entry in enrolled:
        print(f"{entry[0]} {entry[1]} - Enrolled on: {entry[2]}")
else:
    print("  No enrollments found for this course.")
```

OUTPUT SCREENSHOT :

```
Add Student John Doe
1.John Doe inserted successfully.

2.Add Courses
Course 'Introduction to Programming' inserted.
Course 'Mathematics 101' inserted.

3.Enroll John in 2 Courses
John enrolled in 'Introduction to Programming'.
John enrolled in 'Mathematics 101'.

4.Assign Teacher Sarah Smith to Advanced DB Course
Teacher Sarah Smith added.
Course 'Advanced Database Management' added.
Sarah Smith assigned to CS302.

5. Record Payment for Jane Johnson
Jane Johnson added.
Payment recorded for Jane.

6.Enrollment Report for 'Introduction to Programming'
John Doe - Enrolled on: 2024-05-01

7. Enrollment Report for 'Computer Science 101'
 No enrollments found for this course.
```