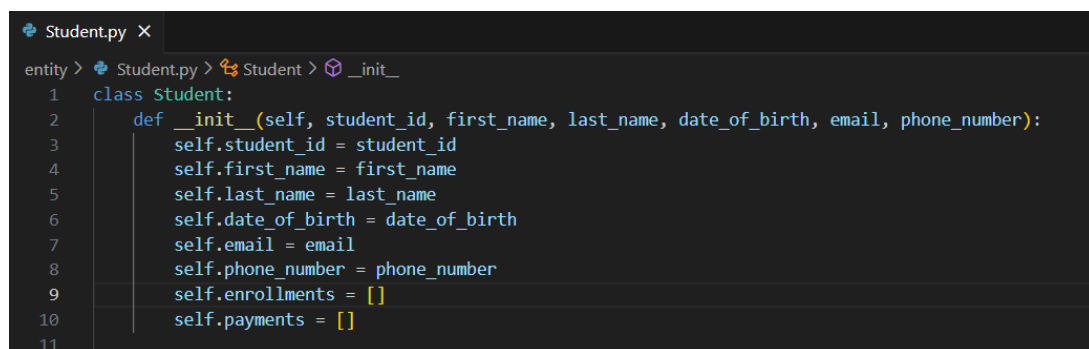


STUDENT INFORMATION SYSTEM – OOPS IMPLEMENTATION

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 = []
```

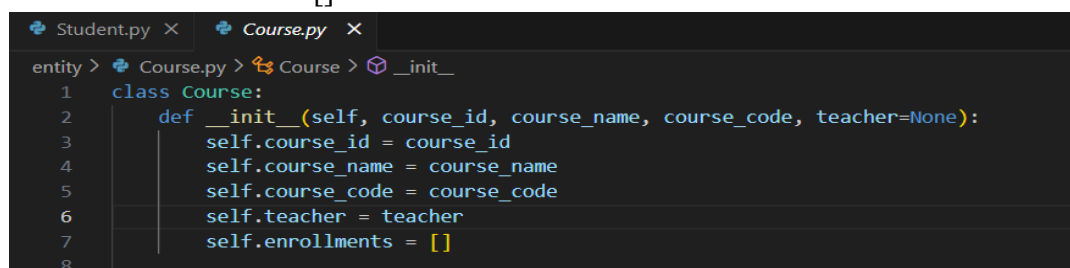


The screenshot shows a code editor with a file named 'Student.py'. The code defines a class 'Student' with an '__init__' method. The method takes six arguments: 'self', 'student_id', 'first_name', 'last_name', 'date_of_birth', 'email', and 'phone_number'. Inside the method, each argument is assigned to a corresponding attribute of 'self'. Finally, two empty lists, 'self.enrollments' and 'self.payments', are initialized.

```
entity > Student.py > Student > __init__
1 class Student:
2     def __init__(self, student_id, first_name, last_name, date_of_birth, email, phone_number):
3         self.student_id = student_id
4         self.first_name = first_name
5         self.last_name = last_name
6         self.date_of_birth = date_of_birth
7         self.email = email
8         self.phone_number = phone_number
9         self.enrollments = []
10        self.payments = []
11
```

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 = []
```



The screenshot shows a code editor with a file named 'Course.py'. The code defines a class 'Course' with an '__init__' method. The method takes four arguments: 'self', 'course_id', 'course_name', 'course_code', and 'teacher' (with a default value of 'None'). Inside the method, each argument is assigned to a corresponding attribute of 'self'. Finally, an empty list, 'self.enrollments', is initialized.

```
entity > Course.py > Course > __init__
1 class Course:
2     def __init__(self, course_id, course_name, course_code, teacher=None):
3         self.course_id = course_id
4         self.course_name = course_name
5         self.course_code = course_code
6         self.teacher = teacher
7         self.enrollments = []
8
```

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__
1 class Enrollment:
2     def __init__(self, enrollment_id, student, course, enrollment_date):
3         self.enrollment_id = enrollment_id
4         self.student = student
5         self.course = course
6         self.enrollment_date = enrollment_date
7
```

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 X Course.py
entity > Student.py > Student > _init_
1 class Student:
11
12     def enroll_in_course(self, course, enrollment_date):
13         from entity.Enrollment import Enrollment
14         enrollment = Enrollment(None, self, course, enrollment_date)
15         self.enrollments.append(enrollment)
16         return enrollment
17
18     def update_info(self, first_name, last_name, date_of_birth, email, phone_number):
19         self.first_name = first_name
20         self.last_name = last_name
21         self.date_of_birth = date_of_birth
22         self.email = email
23         self.phone_number = phone_number
24
25     def make_payment(self, amount, payment_date):
26         from entity.Payment import Payment
27         payment = Payment(None, self, amount, payment_date)
28         self.payments.append(payment)
29         return payment
30
31     def display_info(self):
32         print(f"Student ID: {self.student_id}")
33         print(f"Name: {self.first_name} {self.last_name}")
34         print(f"DOB: {self.date_of_birth}")
35         print(f"Email: {self.email}")
36         print(f"Phone: {self.phone_number}")
37
38     def get_enrolled_courses(self):
39         return [enrollment.course for enrollment in self.enrollments]
40
41     def get_payment_history(self):
42         return self.payments
43
```

2. Course :

```
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
```

```

entity > Course.py > Course > _init_
1 class Course:
2     def __init__(self, course_id, course_name, course_code, teacher=None):
3         self.course_id = course_id
4         self.course_name = course_name
5         self.course_code = course_code
6         self.teacher = teacher
7         self.enrollments = []
8
9     def assign_teacher(self, teacher):
10        self.teacher = teacher
11        teacher.assigned_courses.append(self)
12
13    def update_info(self, course_code, course_name, teacher):
14        self.course_code = course_code
15        self.course_name = course_name
16        self.teacher = teacher
17
18    def display_info(self):
19        print(f"Course ID: {self.course_id}")
20        print(f"Name: {self.course_name}")
21        print(f"Code: {self.course_code}")
22        if self.teacher:
23            print(f"Instructor: {self.teacher.first_name} {self.teacher.last_name}")
24
25    def get_enrollments(self):
26        return self.enrollments
27
28    def get_teacher(self):
29        return self.teacher
30

```

3. Teacher:

```

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

```

```

entity > Teacher.py > ...
1  class Teacher:
2      def __init__(self, teacher_id, first_name, last_name, email):
3          self.teacher_id = teacher_id
4          self.first_name = first_name
5          self.last_name = last_name
6          self.email = email
7          self.assigned_courses = []
8
9      def update_info(self, first_name, last_name, email):
10         self.first_name = first_name
11         self.last_name = last_name
12         self.email = email
13
14         def display_info(self):
15             print(f"Teacher ID: {self.teacher_id}")
16             print(f"Name: {self.first_name} {self.last_name}")
17             print(f"Email: {self.email}")
18
19         def get_assigned_courses(self):
20             return self.assigned_courses
21

```

4. Enrollment:

```

def get_student(self):
    return self.student

def get_course(self):
    return self.course

```

```

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__
1 class Payment:
2     def __init__(self, payment_id, student, amount, payment_date):
3         self.payment_id = payment_id
4         self.student = student
5         self.amount = amount
6         self.payment_date = payment_date
7
8     def get_student(self):
9         return self.student
10
11     def get_payment_amount(self):
12         return self.amount
13
14     def get_payment_date(self):
15         return self.payment_date
16

```

Task 4 - Exceptions handling and Custom Exceptions

```

1 class SISException(Exception):
2     pass
3
4 class DuplicateEnrollmentException(SISException):
5     def __init__(self, message="Student is already enrolled in this course."):
6         super().__init__(message)
7
8 class CourseNotFoundException(SISException):
9     def __init__(self, message="Course not found."):
10        super().__init__(message)
11
12 class StudentNotFoundException(SISException):
13     def __init__(self, message="Student not found."):
14         super().__init__(message)
15
16 class TeacherNotFoundException(SISException):
17     def __init__(self, message="Teacher not found."):
18         super().__init__(message)
19
20 class PaymentValidationException(SISException):
21     def __init__(self, message="Invalid payment details."):
22         super().__init__(message)
23
24 class InvalidStudentDataException(SISException):
25     def __init__(self, message="Invalid student data provided."):
26         super().__init__(message)
27
28 class InvalidCourseDataException(SISException):
29     def __init__(self, message="Invalid course data provided."):
30         super().__init__(message)
31
32 class InvalidEnrollmentDataException(SISException):
33     def __init__(self, message="Invalid enrollment data provided."):
34         super().__init__(message)
35

```

```

35
36 class InvalidTeacherDataException(SISException):
37     def __init__(self, message="Invalid teacher data provided."):
38         super().__init__(message)
39
40 class InsufficientFundsException(SISException):
41     def __init__(self, message="Insufficient funds to enroll in course."):
42         super().__init__(message)
43

```

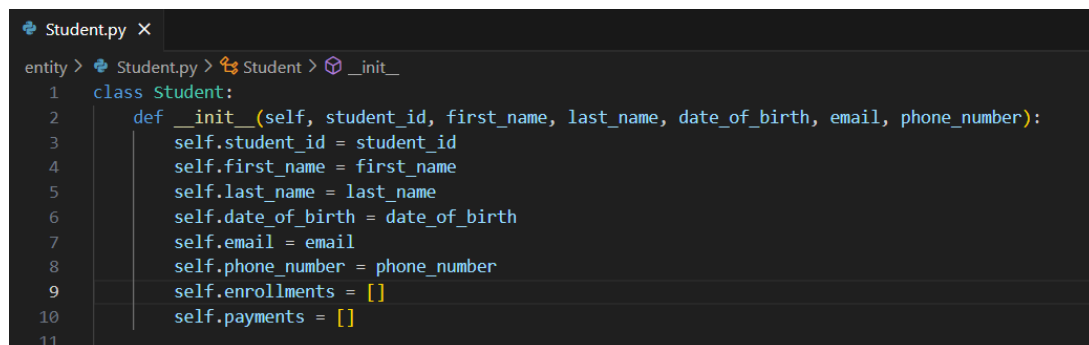
Task 5 – Collections

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 = []

```



The screenshot shows a code editor with a file named 'Student.py'. The code defines a 'Student' class with an '__init__' method. The method takes six arguments: 'student_id', 'first_name', 'last_name', 'date_of_birth', 'email', and 'phone_number'. It assigns these arguments to instance variables 'self.student_id', 'self.first_name', 'self.last_name', 'self.date_of_birth', 'self.email', and 'self.phone_number'. It also initializes two empty lists, 'self.enrollments' and 'self.payments'.

```

Student.py X
entity > Student.py > Student > __init__
1 class Student:
2     def __init__(self, student_id, first_name, last_name, date_of_birth, email, phone_number):
3         self.student_id = student_id
4         self.first_name = first_name
5         self.last_name = last_name
6         self.date_of_birth = date_of_birth
7         self.email = email
8         self.phone_number = phone_number
9         self.enrollments = []
10        self.payments = []
11

```

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__
1 class Course:
2     def __init__(self, course_id, course_name, course_code, teacher=None):
3         self.course_id = course_id
4         self.course_name = course_name
5         self.course_code = course_code
6         self.teacher = teacher
7         self.enrollments = []
8
```

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 = []
```

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__
1 class Enrollment:
2     def __init__(self, enrollment_id, student, course, enrollment_date):
3         self.enrollment_id = enrollment_id
4         self.student = student
5         self.course = course
6         self.enrollment_date = enrollment_date
7
```

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 6

Main Driver Code :

```
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.")

    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}")
```

```

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 :

```

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 :

```
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
```

```
    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
```

```

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

```
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 > ...
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

```
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

```
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

```

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

```

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

```

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.
```