# TASK WISE CODE

NAME: Aniroop Gupta

ASSIGNEMENT: Student Information System

---

**Task 1 and Task 2. Define Classes and Implement Constructor**

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

    def Get_enrollment_id(self):
        return self.__enrollment_id

    def Get_student_id(self):
        return self.__student_id

    def Get_course_id(self):
        return self.__course_id

    def Get_enrollment_date(self):
        return self.__enrollment_date

    def Set_enrollment_id(self, enrollment_id):
        self.__enrollment_id = enrollment_id

    def Set_student_id(self, student_id):
        self.__student_id = student_id

    def Set_course_id(self, course_id):
        self.__course_id = course_id

    def Set_enrollment_date(self, enrollment_date):
        self.__enrollment_date = enrollment_date
```

```python
class Course:
    def __init__(self, course_id, course_name, teacher_id, credits):
        self.course_id = course_id
        self.course_name = course_name
        self.teacher_id = teacher_id
        self.credits = credits

        self.enrollments = []

    def Get_course_id(self):
        return self.__course_id

    def Get_course_name(self):
        return self.__course_name

    def set_course_id(self, course_id):
        self.__course_id = course_id

    def set_course_name(self, course_name):
        self.__course_name = course_name
```

```python
class Payment:
    def __init__(self, payment_id, student_id, amount, payment_date):
        self.__payment_id = payment_id
        self.__student_id = student_id
        self.__amount = amount
        self.__payment_date = payment_date

    def Get_payment_id(self):
        return self.__payment_id

    def Get_student_id(self):
        return self.__student_id

    def Get_amount(self):
        return self.__amount

    def Get_payment_date(self):
        return self.__payment_date

    def Set_payment_id(self, payment_id):
        self.__payment_id = payment_id

    def Set_student_id(self, student_id):
        self.__student_id = student_id

    def Set_amount(self, amount):
        self.__amount = amount

    def Set_payment_date(self, payment_date):
        self.__payment_date = payment_date
```

```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 = []

    def Get_student_id(self):
        return self.__student_id

    def Get_first_name(self):
        return self.__first_name

    def Get_last_name(self):
        return self.__last_name

    def Get_date_of_birth(self):
        return self.__date_of_birth

    def Get_email(self):
        return self.__email

    def Get_phone_number(self):
        return self.__phone_number

    def Set_student_id(self, student_id):
        self.__student_id = student_id

    def Set_first_name(self, first_name):
        self.__first_name = first_name

    def Set_last_name(self, last_name):
        self.__last_name = last_name
```

```python
    def Set_last_name(self, last_name):
        self.__last_name = last_name

    def Set_date_of_birth(self, date_of_birth):
        self.__date_of_birth = date_of_birth

    def Set_email(self, email):
        self.__email = email

    def Set_phone_number(self, phone_number):
        self.__phone_number = phone_number
```

```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.courses_assigned = []

    def Get_teacher_id(self):
        return self.__teacher_id

    def Get_first_name(self):
        return self.__first_name

    def Get_last_name(self):
        return self.__last_name

    def Get_email(self):
        return self.__email

    def Set_teacher_id(self, teacher_id):
        self.__teacher_id = teacher_id

    def Set_first_name(self, first_name):
        self.__first_name = first_name

    def Set_last_name(self, last_name):
        self.__last_name = last_name

    def Set_email(self, email):
        self.__email = email
```

```python
class SIS:
    def __init__(self):
        self.students = []
        self.courses = []
        self.teachers = []
        self.enrollments = []
        self.payments = []
```

## Task 3. Implement Methods

```
class StudentServiceImpl(StudentDAO, DBConnection):

    def Add_student(self):
        first_name = input("Enter first name: ")
        last_name = input("Enter last name: ")
        date_of_birth = input("Enter date of birth (YYYY-MM-DD): ")
        email = input("Enter email: ")
        phone_number = input("Enter phone number: ")
        if not first_name or not last_name or not date_of_birth or not email or not phone_number:
            raise InvalidStudentDataException("All fields are required.")

        with self.conn:
            self.conn.execute(
                "INSERT INTO students (first_name, last_name, date_of_birth, email, phone_number) VALUES (?, ?, ?, ?, ?)",
                (first_name, last_name, date_of_birth, email, phone_number))
        print("Student added successfully!")

    def Update_student(self):
        student_id = input("Enter the student ID to update: ")
        first_name = input("Enter updated first name: ")
        last_name = input("Enter updated last name: ")
        date_of_birth = input("Enter updated date of birth (YYYY-MM-DD): ")
        email = input("Enter updated email: ")
        phone_number = input("Enter updated phone number: ")

        with self.conn:
            self.conn.execute(
                "UPDATE students SET first_name=?, last_name=?, date_of_birth=?, email=?, phone_number=? WHERE student_id=?",
                (first_name, last_name, date_of_birth, email, phone_number, student_id))
        print("Student updated successfully!")

    def Get_student(self):
        try:
            student_id = int(input("Enter student_id: "))
            print("Searching for student with ID:", student_id)

            cursor = self.conn.cursor()
            cursor.execute("SELECT * FROM students WHERE student_id = ?", (student_id,))
            row = cursor.fetchone()

            if row:
                student = Student(*row)
                print("Student found:", student)
                return student
            else:
                raise StudentNotFoundException(f"No student found with ID: {student_id}")

        except StudentNotFoundException as e:
            print(e)
        except Exception as e:
            print("Error retrieving student:", e)
        finally:
            cursor.close()

    def Delete_student(self):
        student_id = int(input("Enter student_id: "))

        with self.conn:
            self.conn.execute("DELETE FROM students WHERE student_id= ?", (student_id,))
        print("Student deleted successfully!")

    def Get_all_students(self):
        try:
            cursor = self.conn.cursor()
            cursor.execute("SELECT * FROM students")
            students = [Student(*row) for row in cursor.fetchall()]
            cursor.close()

            if students:
                print("All students:")
                for student in students:
                    print(f"Student ID: {student.student_id}")
```

```
                print(f"First Name: {student.first_name}")
                print(f"Last Name: {student.last_name}")
                print(f"Date of Birth: {student.date_of_birth}")
                print(f"Email: {student.email}")
                print(f"Phone Number: {student.phone_number}")
                print()
            else:
                print("No students found.")
        except Exception as e:
            print("Error retrieving students:", str(e))


lass CourseServiceImpl(CourseDAO, DBConnection):

    def Add_course(self):
        course_id = int(input("Enter course_id: "))
        course_name = input("Enter course name: ")
        teacher_id = int(input("Enter teacher_id: "))
        credits = int(input("Enter credits: "))
        if not course_name or credits <= 0:
            raise InvalidCourseDataException("Course name is required and credits should be a positive number.")

        with self.conn:
            self.conn.execute("INSERT INTO courses (course_id, course_name, teacher_id, credits) VALUES (?, ?, ?, ?)",
                              (course_id, course_name, teacher_id, credits))
        print("Course added successfully!")

    def Update_course(self):
        course_id = int(input("Enter course_id: "))
        course_name = input("Enter updated course name: ")
        teacher_id = int(input("Enter updated teacher_id: "))
        credits = int(input("Enter updated credits: "))

        with self.conn:
            self.conn.execute("UPDATE courses SET course_name=?, teacher_id=?, credits=? WHERE course_id=?",
                              (course_name, teacher_id, credits, course_id))
        print("Course updated successfully!")

    def Get_course(self):
        try:
            course_id = int(input("Enter course_id: "))
            print("Searching for course with ID:", course_id)

            cursor = self.conn.cursor()
            cursor.execute("SELECT * FROM courses WHERE course_id = ?", (course_id,))
            row = cursor.fetchone()
            cursor.close()

            if row:
                course = Course(*row)
                print(f"Course ID: {course.course_id}")
                print(f"Course Name: {course.course_name}")
                print(f"Credits: {course.credits}")
                print(f"Teacher ID: {course.teacher_id}")
            else:
                raise CourseNotFoundException(f"No course found with ID: {course_id}")

        except CourseNotFoundException as e:
            print(e)
        except Exception as e:
            print("Error retrieving course:", e)

    def Delete_course(self):
        course_id = int(input("Enter course_id: "))

        with self.conn:
            self.conn.execute("DELETE FROM courses WHERE course_id= ?", (course_id,))
        print("Course deleted successfully!")
```

```
    def Get_all_courses(self):
        try:
            cursor = self.conn.cursor()
            cursor.execute("SELECT * FROM courses")
            courses = [Course(*row) for row in cursor.fetchall()]
            cursor.close()

            if courses:
                print("All courses:")
                for course in courses:
                    print(f"Course ID: {course.course_id}")
                    print(f"Course Name: {course.course_name}")
                    print(f"Credits: {course.credits}")
                    print(f"Teacher ID: {course.teacher_id}")
                    print()
            else:
                print("No courses found.")
        except Exception as e:
            print("Error retrieving courses:", str(e))

from typing import List

class EnrollmentServiceImpl(EnrollmentDAO, DBConnection):
    def Add_enrollment(self):
        try:
            # Input values for enrollment
            student_id = int(input("Enter student_id: "))
            course_id = int(input("Enter course_id: "))
            enrollment_date = input("Enter enrollment date: ")

            if not enrollment_date:
                raise InvalidEnrollmentDataException("Enrollment date is required.")

            # Create a cursor from the connection
            cursor = self.conn.cursor()
```

```
            # Check if the student is already enrolled in the course
            cursor.execute("SELECT * FROM Enrollments WHERE student_id = ? AND course_id = ?", (student_id, course_id))
            existing = cursor.fetchone()

            if existing:
                raise DuplicateEnrollmentException("Student is already enrolled in the course.")

            # Insert new enrollment if no existing record is found
            cursor.execute(
                "INSERT INTO Enrollments (student_id, course_id, enrollment_date) VALUES (?, ?, ?)",
                (student_id, course_id, enrollment_date)
            )

            # Commit the transaction
            self.conn.commit()

            # Close the cursor
            cursor.close()

            print("Enrollment added successfully!")

        except InvalidEnrollmentDataException as e:
            print(f"Error: {e}")
        except DuplicateEnrollmentException as e:
            print(f"Error: {e}")
        except Exception as e:
            print(f"An error occurred: {e}")

    def Update_enrollment(self):
        student_id = int(input("Enter student_id: "))
        course_id = int(input("Enter course_id: "))
        enrollment_date = input("Enter enrollment date: ")
        enrollment_id = int(input("Enter enrollment id: "))

        self.conn.execute("UPDATE Enrollments SET student_id = ?, course_id = ?, enrollment_date = ? WHERE enrollment_id = ?",
```

```
                    (student_id, course_id, enrollment_date, enrollment_id))
        self.conn.commit()
        print("Enrollment updated successfully!")

    def Get_enrollment(self):
        course_id = int(input("Enter Course ID: "))
        cursor = self.conn.cursor()
        cursor.execute("SELECT * FROM Enrollments WHERE course_id = ?", (course_id,))
        rows = cursor.fetchall()  # Fetch all results
        cursor.close()

        if rows:
            for row in rows:
                enrollment_id, student_id, course_id, enrollment_date = row
                print(f"Enrollment ID: {enrollment_id}")
                print(f"Student ID: {student_id}")
                print(f"Course ID: {course_id}")
                print(f"Enrollment Date: {enrollment_date}")
                print()  # Add a new line for better readability
        else:
            print("No enrollments found for Course ID:", course_id)

    def Delete_enrollment(self):
        enrollment_id = int(input("Enter enrollment id: "))
        self.conn.execute("DELETE FROM Enrollments WHERE enrollment_id = ?", (enrollment_id,))
        self.conn.commit()
        print("Enrollment deleted successfully!")

    def Get_all_enrollments(self) -> List[Enrollment]:
        cursor = self.conn.cursor()
        cursor.execute("SELECT * FROM Enrollments")
        enrollments = [Enrollment(*row) for row in cursor.fetchall()]
        cursor.close()

        if enrollments:
            for enrollment in enrollments:
                print(f"Enrollment ID: {enrollment.Get_enrollment_id()}")
                print(f"Student ID: {enrollment.student_id}")
                print(f"Course ID: {enrollment.course_id}")
                print(f"Enrollment Date: {enrollment.enrollment_date}")
        else:
            print("No enrollments found.")
        return enrollments


lass TeacherServiceImpl(TeacherDAO, DBConnection):

    def Add_teacher(self):
        first_name = input("Enter first name: ")
        last_name = input("Enter last name: ")
        email = input("Enter email: ")

        if not first_name or not last_name or not email:
            raise InvalidTeacherDataException("First name, last name, and email are required.")

        self.conn.execute("INSERT INTO teacher (first_name, last_name, email) VALUES (?, ?, ?)",
                          (first_name, last_name, email))
        self.conn.commit()
        print("Teacher added successfully!")

    def Update_teacher(self):
        first_name = input("Enter first name: ")
        last_name = input("Enter last name: ")
        email = input("Enter email: ")
        teacher_id = int(input("Enter teacher id: "))

        self.conn.execute("UPDATE teacher SET first_name = ?, last_name = ?, email = ? WHERE teacher_id = ?",
                          (first_name, last_name, email, teacher_id))
        self.conn.commit()
        print("Teacher updated successfully!")
```

```
    def Get_teacher(self):
        teacher_id = int(input("Enter teacher id: "))
        cursor = self.conn.cursor()
        cursor.execute("SELECT * FROM teacher WHERE teacher_id = ?", (teacher_id,))
        row = cursor.fetchone()
        cursor.close()

        if row:
            teacher_id, first_name, last_name, email = row
            print(f"Teacher ID: {teacher_id}")
            print(f"First Name: {first_name}")
            print(f"Last Name: {last_name}")
            print(f"Email: {email}")
        else:
            print("No teacher found with ID:", teacher_id)

    def Delete_teacher(self):
        teacher_id = int(input("Enter teacher id: "))
        self.conn.execute("DELETE FROM teacher WHERE teacher_id = ?", (teacher_id,))
        self.conn.commit()
        print("Teacher deleted successfully!")

    def Get_all_teachers(self) -> List[Teacher]:
        cursor = self.conn.cursor()
        cursor.execute("SELECT * FROM teacher")
        teachers = [Teacher(*row) for row in cursor.fetchall()]
        cursor.close()

        if teachers:
            for teacher in teachers:
                print(f"Teacher ID: {teacher.teacher_id}")
                print(f"First Name: {teacher.first_name}")
                print(f"Last Name: {teacher.last_name}")
                print(f"Email: {teacher.email}")
        else:
            print("No teachers found.")
        return teachers


class PaymentServiceImpl(PaymentDAO, DBConnection):

    def Add_payment(self):
        payment_id = int(input("Enter payment id: "))
        student_id = int(input("Enter student id: "))
        amount = int(input("Enter amount: "))
        payment_date = input("Enter payment_date: ")

        if amount <= 0:
            raise PaymentValidationException("Amount must be positive.")
        if student_id <= 0:
            raise PaymentValidationException("Student ID must be positive.")

        self.conn.execute("INSERT INTO payments (payment_id, student_id, amount, payment_date) VALUES (?, ?, ?, ?)",
                          (payment_id, student_id, amount, payment_date))
        self.conn.commit()
        print("Payment added successfully!")

    def Update_payment(self):
        student_id = int(input("Enter student_id: "))
        amount = int(input("Enter amount: "))
        payment_date = input("Enter date: ")
        payment_id = int(input("Enter payment ID: "))

        self.conn.execute("UPDATE payments SET student_id = ?, amount = ?, payment_date = ? WHERE payment_id = ?",
                          (student_id, amount, payment_date, payment_id))
        self.conn.commit()
        print("Payment updated successfully!")

    def Get_payment(self):
        payment_id = int(input("Enter payment ID: "))
        cursor = self.conn.cursor()
```

(variable) payment_date: str

```
        cursor.execute("SELECT * FROM payments WHERE payment_id = ?", (payment_id,))
        row = cursor.fetchone()
        cursor.close()

        if row:
            payment_id, student_id, amount, payment_date = row
            print(f"Payment ID: {payment_id}")
            print(f"Student ID: {student_id}")
            print(f"Amount: {amount}")
            print(f"Payment Date: {payment_date}")
        else:
            print("No payment found with ID:", payment_id)

    def Delete_payment(self):
        payment_id = int(input("Enter payment id: "))
        self.conn.execute("DELETE FROM payments WHERE payment_id = ?", (payment_id,))
        self.conn.commit()
        print("Payment deleted successfully!")

    def Get_all_payments(self) -> List[Payment]:
        cursor = self.conn.cursor()
        cursor.execute("SELECT * FROM payments")
        payments = [Payment(*row) for row in cursor.fetchall()]
        cursor.close()

        if payments:
            for payment in payments:
                print(f"Payment ID: {payment.Get_payment_id()}")
                print(f"Student ID: {payment.Get_student_id()}")
                print(f"Amount: {payment.Get_amount()}")
                print(f"Payment Date: {payment.Get_payment_date()}")
        else:
            print("No payments found.")
        return payments
```

**Task 4: Exceptions handling and Custom Exceptions**

```python
class DuplicateEnrollmentException(Exception):
    def __init__(self, message="Duplicate enrollment detected"):
        self.message = message
        super().__init__(self.message)


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


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


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


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


class InvalidDataException(Exception):
    def __init__(self, message="Invalid data provided"):
        self.message = message
        super().__init__(self.message)
```

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


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


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


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


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

**Task 5: Collections**

```python
class Course:
    def __init__(self, course_id, course_name, teacher_id, credits):
        self.course_id = course_id
        self.course_name = course_name
        self.teacher_id = teacher_id
        self.credits = credits

        self.enrollments = []
```

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

```python
class Payment:
    def __init__(self, payment_id, student_id, amount, payment_date):
        self.__payment_id = payment_id
        self.__student_id = student_id
        self.__amount = amount
        self.__payment_date = payment_date
```

```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 = []
```

```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.courses_assigned = []
```

## Task 6: Create Methods for Managing Relationships (Driver Method)

```
class Main:
    def _init_(self):
        self.loop = None

    def main(self):
        self.loop = True
        while self.loop:
            try:
                student_service = StudentServiceImpl()
                course_service = CourseServiceImpl()
                enrollment_service = EnrollmentServiceImpl()
                teacher_service = TeacherServiceImpl()
                payment_service= PaymentServiceImpl()
                print("Welcome to  the Student Information System")
                print("Select option to use functionalities: ")
                print("1.Student\n2.Course\n3.Enrollment\n4.Teacher\n5.Payment\n6.Exit")
                choice = int(input("Enter the choice: "))
                if choice in range(1, 7):
                    if choice == 1:
                        while True:
                            print('''1.Enroll a new student\n2.Update student\n3.Get Student\n4.Delete Student\n5.Get all students\n6.Exit''')
                            choice_1 = int(input("Enter your Choice: "))
                            if choice_1 in range(1, 7):
                                if choice_1 == 1:
                                    student_service.Add_student()
                                elif choice_1 == 2:
                                    student_service.Update_student()
                                elif choice_1 == 3:
                                    student_service.Get_student()
                                elif choice_1 == 4:
                                    student_service.Delete_student()
                                elif choice_1 == 5:
                                    student_service.Get_all_students()
                                else:
                                    break
                            else:
                                raise InvalidDataException("Input should be between 1 and 6")
```

```
                elif choice == 2:
                    while True:
                        print('''1.Add course\n2.Update course\n3.Get course\n4.Delete course\n5.Get all courses\n6.Exit''')
                        choice_2 = int(input("Enter your Choice: "))
                        if choice_2 in range(1, 7):
                            if choice_2 == 1:
                                course_service.Add_course()
                            elif choice_2 == 2:
                                course_service.Update_course()
                            elif choice_2 == 3:
                                course_service.Get_course()
                            elif choice_2 == 4:
                                course_service.Delete_course()
                            elif choice_2 == 5:
                                course_service.Get_all_courses()
                            else:
                                break
                        else:
                            raise InvalidDataException("Input should be between 1 and 6")
                elif choice == 3:
                    while True:
                        print('''1.Add enrollments\n2.Update enrollments\n3.Get enrollments\n4.Delete enrollments\n5.Get all enrollments\n6.
                        Exit''')
                        choice_3 = int(input("Enter your Choice: "))
                        if choice_3 in range(1, 7):
                            if choice_3 == 1:
                                enrollment_service.Add_enrollment()
                            elif choice_3 == 2:
                                enrollment_service.Update_enrollment()
                            elif choice_3 == 3:
                                enrollment_service.Get_enrollment()
                            elif choice_3 == 4:
                                enrollment_service.Delete_enrollment()
                            elif choice_3 == 5:
                                enrollment_service.Get_all_enrollments()
```

```
                        else:
                            break
                    else:
                        raise InvalidDataException("Input should be between 1 and 6")
                elif choice == 4:
                    while True:
                        print('''1.Add teacher\n2.Update teacher\n3.Get teacher\n4.Delete teacher\n5.Get all teachers\n6.Exit''')
                        choice_4 = int(input("Enter your Choice: "))
                        if choice_4 in range(1, 7):
                            if choice_4 == 1:
                                teacher_service.Add_teacher()
                            elif choice_4 == 2:
                                teacher_service.Update_teacher()
                            elif choice_4 == 3:
                                teacher_service.Get_teacher()
                            elif choice_4 == 4:
                                teacher_service.Delete_teacher()
                            elif choice_4 == 5:
                                teacher_service.Get_all_teachers()
                            else:
                                break
                        else:
                            raise InvalidDataException("Input should be between 1 and 6")
                elif choice == 5:
                    while True:
                        print('''1.Add payment\n2.Update payment\n3.Get payment\n4.Delete payment\n5.Get all payments\n6.Exit''')
                        choice_5 = int(input("Enter your Choice: "))
                        if choice_5 in range(1, 7):
                            if choice_5 == 1:
                                payment_service.Add_payment()
                            elif choice_5 == 2:
                                payment_service.Update_payment()
                            elif choice_5 == 3:
                                payment_service.Get_payment()
                            elif choice_5 == 4:
                                payment_service.Delete_payment()
                            elif choice_5 == 5:
                                payment_service.Get_all_payments()
                            else:
                                break
                        else:
                            raise InvalidDataException("Input should be between 1 and 6")
                else:
                    exit()
            except Exception as e:
                print(f"An error occurred: {e}")
            finally:
                print("Thank You for reaching Student Information System...")


obj = Main()
obj.main()
```

```
Welcome to  the Student Information System
Select option to use functionalities:
1.Student
2.Course
3.Enrollment
4.Teacher
5.Payment
6.Exit
Enter the choice:
```

**Task 7: Database Connectivity**

```python
class PropertyUtil:
    @staticmethod
    def get_property_string():
        server_name = "LAPTOP-7MH0675Q\SQLEXPRESS01"
        database_name = "SISDB"

        conn_str = (
            f"Driver={{SQL Server}};"
            f"Server={server_name};"
            f"Database={database_name};"
            f"Trusted_Connection=yes;"
        )

        return conn_str
```

```python
import pyodbc
from util.dbconnection import PropertyUtil

class DBConnection:
    def __init__(self):
        conn_str = PropertyUtil.get_property_string()
        self.conn = pyodbc.connect(conn_str)
        self.cursor = self.conn.cursor()

    def close(self):
        self.cursor.close()
        self.conn.close()
```

**Task 8: Student Enrollment**

```
Enter your Choice: 1
Enter first name: John
Enter last name: Doe
Enter date of birth (YYYY-MM-DD): 1995-08-15
Enter email: john.doe@example.com
Enter phone number: 123-456-7890
student added successfully!
```

Results Messages

| | student_id | first_name | last_name | date_of_birth | email | phone_number |
|---|---|---|---|---|---|---|
| 1 | 11 | John | Doe | 1995-08-15 | john.doe@example.com | 123-456-7890 |

```
Enter your Choice: 1
Enter student_id: 11
Enter course_id: 11
Enter enrollment date: 2023-11-11
Enrollment added successfully!
```

```
Enter your Choice: 1
Enter student_id: 11
Enter course_id: 12
Enter enrollment date: 2023-11-11
Enrollment added successfully!
```

Results Messages

| | enrollment_id | student_id | course_id | enrollment_date |
|---|---|---|---|---|
| 1 | 12 | 11 | 11 | 2023-11-11 |
| 2 | 13 | 11 | 12 | 2023-11-11 |

**Task 9: Teacher Assignment**

```
Enter your Choice: 2
Enter course_id: 13
Enter updated course name: Advanced Database Management
Enter updated teacher_id: 21
Enter updated credits: 4
Course updated successfully!
```

Results Messages

| | teacher_id | first_name | last_name | email |
|---|---|---|---|---|
| 1 | 21 | Sarah | Smith | sarah.smith@example.com |

Results Messages

| | course_id | course_name | credits | teacher_id |
|---|---|---|---|---|
| 1 | 13 | Advanced Database Management | 4 | 21 |

**Task 10: Payment Record**

```
Enter your Choice: 1
Enter payment id: 11
Enter student id: 101
Enter amount: 500
Enter payment_date: 2023-04-10
Payment added successfully!
```

Results Messages

| | student_id | first_name | last_name | date_of_birth | email | phone_number |
|---|---|---|---|---|---|---|
| 1 | 101 | Jane | Johnson | 1999-11-12 | jane@example.com | 258744166 |

Results Messages

| | payment_id | student_id | amount | payment_date |
|---|---|---|---|---|
| 1 | 11 | 101 | 500 | 2023-04-10 |

**Task 11: Enrollment Report Generation**

```
Enter your Choice: 3
Enter Course ID: 14
Enrollment ID: 14
Student ID: 83
Course ID: 14
Enrollment Date: 2023-05-11

Enrollment ID: 15
Student ID: 45
Course ID: 14
Enrollment Date: 2023-04-08
```

Results Messages

| | course_id | course_name | credits | teacher_id |
|---|---|---|---|---|
| 1 | 14 | Computer Science 101 | 5 | 2 |

Results Messages

| | enrollment_id | student_id | course_id | enrollment_date |
|---|---|---|---|---|
| 1 | 14 | 83 | 14 | 2023-05-11 |
| 2 | 15 | 45 | 14 | 2023-04-08 |