Submitted By : Mridul Bhardwaj

Assignment -2

Student Information System (SIS)

Implement OOPs

A Student Information System (SIS) manages information about students, courses, student enrollments, teachers, and payments. Each student can enroll in multiple courses, each course can have multiple students, each course is taught by a teacher, and students make payments for their courses. Students have attributes such as name, date of birth, email, and phone number. Courses have attributes such as course name, course code, and instructor name. Enrollments track which students are enrolled in which courses. Teachers have attributes such as names and email. Payments track the amount and date of payments made by students.

Task 1:

Define Classes Define the following classes based on the domain description:

Student class with the following attributes:

- Student ID
- First Name
- Last Name
- Date of Birth
- Email
- Phone Number

```python
from DBconnection import DBConnection

class student(DBConnection):
    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
```

Course class with the following attributes:

- Course ID
- Course Name
- Course Code

• Instructor Name

```
class course :
    def __init__(self, course_id, course_name, course_code, instructor_name):
        self.course_id= course_id
        self.course_name= course_name
        self.course_code= course_code
        self.instructor_name= instructor_name
```

Enrollment class to represent the relationship between students and courses.

It should have attributes:

• Enrollment ID

• Student ID (reference to a Student)

• Course ID (reference to a Course)

• Enrollment Date

```
from DBconnection import DBConnection

class Enrollment(DBConnection):
    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
```

Teacher class with the following attributes:

• Teacher ID

• First Name

• Last Name

• Email

```
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
```

Payment class with the following attributes:

• Payment ID

• Student ID (reference to a Student)

• Amount

• Payment Date

```
from DBconnection import DBConnection
class Payment(DBConnection):
    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
```

Task 3:

Implement Methods Implement methods in your classes to perform various operations related to the Student Information System (SIS). These methods will allow you to interact with and manipulate data within your system.

Below are detailed instructions on how to implement methods in each class:

Implement the following methods in the appropriate classes:

Student Class:

• EnrollInCourse(course: Course): Enrolls the student in a course.

```
def EnrollInCourse(self):
    try:
        self.open()
        self.c.execute(f"INSERT INTO Enrollments (enrollment_id,student_id, course_id, enrollment
        self.mydb.commit()
        print("Enrolled student in the course successfully.")
    except Exception as e:
        print("Error while enrolling student in course:", e)
    finally:
        self.close()
```

• UpdateStudentInfo(firstName: string, lastName: string, dateOfBirth: DateTime, email: string, phoneNumber: string): Updates the student's information.

```python
class Repository(DBConnection):
    def __init__(self):
        pass
    def UpdateStudentInfo(self, student_id, new_first_name, new_last_name, new_date_of_birth,
                          new_email, new_phone_number):
        try:
            self.open()
            self.c.execute(f"UPDATE students SET first_name = ('{new_first_name}'),
                           last_name = ('{new_last_name}'),
                           date_of_birth = ('{new_date_of_birth}'),
                           email = ('{new_email}'),phonenumber = ('{new_phone_number}')
                           WHERE student_id =('{student_id}')")
            self.mydb.commit()
            print("Student information updated successfully.")
        except Exception as e:
            print("Error while updating student information:", e)
        finally:
            self.close()
```

• MakePayment(amount: decimal, paymentDate: DateTime): Records a payment made by the student.

```python
def MakePayment(self):
    try:
        self.open()
        self.c.execute(f" INSERT INTO Payments (payment_id,student_id, amount, payment_date) VALU
        self.mydb.commit()
        print("Recorded payment successfully.")
    except Exception as e:
        print("Error while recording payment:", e)
    finally:
        self.close()
```

• DisplayStudentInfo(): Displays detailed information about the student.

```python
def DisplayStudentInfo(self, student_id):
    try:
        self.open()
        query = f"""
            SELECT *
            FROM students
            WHERE student_id = {student_id}
        """

        self.c.execute(query)
        student_info = self.c.fetchone()

        if student_info:
            print("Student Information:")
            print(f"Student ID: {student_info[0]}")
            print(f"First Name: {student_info[1]}")
            print(f"Last Name: {student_info[2]}")
            print(f"Date of Birth: {student_info[3]}")
            print(f"Email: {student_info[4]}")
            print(f"Phone Number: {student_info[5]}")
        else:
            print("Student not found.")
    except Exception as e:
        print("Error while displaying student information:", e)
    finally:
        self.close()
```

• GetEnrolledCourses(): Retrieves a list of courses in which the student is enrolled.

```python
def GetEnrolledCourses(self, student_id):
    try:
        self.open()
        query = f"""
            SELECT c.course_name
            FROM courses c
            INNER JOIN enrollments e ON c.course_id = e.course_id
            WHERE e.student_id = {student_id}
        """

        self.c.execute(query)
        enrolled_courses = self.c.fetchall()

        if enrolled_courses:
            # Return the list of enrolled courses
            return enrolled_courses
        else:
            return []  # Return an empty list if no courses are enrolled
    except Exception as e:
        print("Error while retrieving enrolled courses:", e)
        return []  # Return an empty list in case of an error
    finally:
        self.close()
```

• GetPaymentHistory(): Retrieves a list of payment records for the student.

```
def GetPaymentHistory(self, student_id):
    try:
        self.open()
        query = f"""
            SELECT amount, payment_date
            FROM payments
            WHERE student_id = {student_id}
        """

        self.c.execute(query)
        payment_history = self.c.fetchall()

        return payment_history
    except Exception as e:
        print("Error while retrieving payment history:", e)
        return []
    finally:
        self.close()
```

Course Class:

• AssignTeacher(teacher: Teacher): Assigns a teacher to the course.

```
def AssignTeacher(self, course_id, teacher_id):
    try:
        self.open()
        query = f"""
            UPDATE courses
            SET teacher_id = {teacher_id}
            WHERE course_id = {course_id}
        """

        self.c.execute(query)
        self.mydb.commit()
        print("Teacher assigned to the course successfully.")
    except Exception as e:
        print("Error while assigning teacher to course:", e)
    finally:
        self.close()
```

• UpdateCourseInfo(courseCode: string, courseName: string, instructor: string): Updates course information.

```
def UpdateCourseInfo(self, course_name, credits, teacher_id):
    try:
        self.open()
        query = f"""
            UPDATE courses
            SET credits = {credits},
                teacher_id = {teacher_id}
            WHERE course_name = '{course_name}'
        """

        self.c.execute(query)
        self.mydb.commit()
        print("Course information updated successfully.")
    except Exception as e:
        print("Error while updating course information:", e)
    finally:
        self.close()
```

• DisplayCourseInfo(): Displays detailed information about the course.

```
def DisplayCourseInfo(self, course_name):
    try:
        self.open()
        query = f"""
            SELECT *
            FROM courses
            WHERE course_name = '{course_name}'
        """

        self.c.execute(query)
        course_info = self.c.fetchone()

        if course_info:
            print("Course Information:")
            print(f"Course ID: {course_info[0]}")
            print(f"Course Name: {course_info[1]}")
            print(f"Credits: {course_info[2]}")
            print(f"Teacher ID: {course_info[3]}")
        else:
            print("Course not found.")
    except Exception as e:
        print("Error while displaying course information:", e)
    finally:
        self.close()
```

• GetEnrollments(): Retrieves a list of student enrollments for the course.

```python
def GetEnrollments(self, course_name):
    try:
        self.open()
        query = f"""
            SELECT s.first_name, s.last_name
            FROM students s
            INNER JOIN enrollments e ON s.student_id = e.student_id
            INNER JOIN courses c ON c.course_id = e.course_id
            WHERE c.course_name = '{course_name}'
        """

        self.c.execute(query)
        student_enrollments = self.c.fetchall()

        if student_enrollments:
            print("Student Enrollments for the Course:")
            for student in student_enrollments:
                print(f"Student: {student[0]} {student[1]}")
        else:
            print("No student enrollments found for this course.")
    except Exception as e:
        print("Error while retrieving student enrollments:", e)
    finally:
        self.close()
```

• GetTeacher(): Retrieves the assigned teacher for the course.

```
def GetTeacher(self, course_name):
    try:
        self.open()
        query = f"""
            SELECT t.first_name, t.last_name
            FROM teacher t
            INNER JOIN courses c ON c.teacher_id = t.teacher_id
            WHERE c.course_name = '{course_name}'
        """

        self.c.execute(query)
        teacher_info = self.c.fetchone()

        if teacher_info:
            print("Assigned Teacher for the Course:")
            print(f"Teacher: {teacher_info[0]} {teacher_info[1]}")
        else:
            print("No teacher assigned for this course.")
    except Exception as e:
        print("Error while retrieving assigned teacher:", e)
    finally:
        self.close()
```

Enrollment Class:

- GetStudent(): Retrieves the student associated with the enrollment.

```
def GetStudent(self, enrollment_id):
    try:
        self.open()
        query = f"""
            SELECT s.first_name, s.last_name
            FROM students s
            INNER JOIN enrollments e ON s.student_id = e.student_id
            WHERE e.enrollment_id = {enrollment_id}
        """

        self.c.execute(query)
        student_info = self.c.fetchone()

        if student_info:
            print("Student Associated with the Enrollment:")
            print(f"Student: {student_info[0]} {student_info[1]}")
        else:
            print("No student associated with this enrollment.")
    except Exception as e:
        print("Error while retrieving student information:", e)
    finally:
        self.close()
```

• GetCourse(): Retrieves the course associated with the enrollment.

```
def GetCourse(self, enrollment_id):
    try:
        self.open()
        query = f"""
            SELECT c.course_name
            FROM courses c
            INNER JOIN enrollments e ON c.course_id = e.course_id
            WHERE e.enrollment_id = {enrollment_id}
        """

        self.c.execute(query)
        course_info = self.c.fetchone()

        if course_info:
            print("Course Associated with the Enrollment:")
            print(f"Course Name: {course_info[0]}")
        else:
            print("No course associated with this enrollment.")
    except Exception as e:
        print("Error while retrieving course information:", e)
    finally:
        self.close()
```

Teacher Class:

• UpdateTeacherInfo(name: string, email: string, expertise: string): Updates teacher information.

```
def UpdateTeacherInfo(self, teacher_id, email,first_name,last_name):
    try:
        self.open()
        query = f"""
            UPDATE teacher
            SET email = '{email}',first_name='{first_name}',last_name='{last_name}'
            WHERE teacher_id = {teacher_id}
        """

        self.c.execute(query)
        self.mydb.commit()
        print("Teacher information updated successfully.")
    except Exception as e:
        print("Error while updating teacher information:", e)
    finally:
        self.close()
```

• DisplayTeacherInfo(): Displays detailed information about the teacher.

```
def DisplayTeacherInfo(self, teacher_id):
    try:
        self.open()
        query = f"""
            SELECT *
            FROM teacher
            WHERE teacher_id = {teacher_id}
        """

        self.c.execute(query)
        teacher_info = self.c.fetchone()

        if teacher_info:
            print("Teacher Information:")
            print(f"Teacher ID: {teacher_info[0]}")
            print(f"First Name: {teacher_info[1]}")
            print(f"Last Name: {teacher_info[2]}")
            print(f"Email: {teacher_info[3]}")
        else:
            print("Teacher not found.")
    except Exception as e:
        print("Error while displaying teacher information:", e)
    finally:
        self.close()
```

• GetAssignedCourses(): Retrieves a list of courses assigned to the teacher.

```
def GetAssignedCourses(self, teacher_id):
    try:
        self.open()
        query = f"""
            SELECT c.course_name
            FROM courses c
            WHERE c.teacher_id = {teacher_id}
        """

        self.c.execute(query)
        assigned_courses = self.c.fetchall()

        if assigned_courses:
            print("Courses Assigned to the Teacher:")
            for course in assigned_courses:
                print(f"Course Name: {course[0]}")
        else:
            print("No courses assigned to this teacher.")
    except Exception as e:
        print("Error while retrieving assigned courses:", e)
    finally:
        self.close()
```

Implement the Main Method

In the console application, the Main method serves as the entry point for your program. This is where you will create instances of your classes, call methods, and interact with your Student Information System.

In the Main method, you create instances of your classes (e.g., Student, Course, and SIS) and then interact with your Student Information System by calling methods and handling exceptions.

```
from Enrollment import Enrollment
from Payment import Payment
from service import Repository

class Main():
    def main():
        r=Repository()
        while True:
            print("\n----------Main Menu----------")
            print("\nPress-1 Enroll in course")
            print("Press-2 Update student info")
            print("Press-3 Record payment")
            print("Press-4 Get student info")
            print("Press-5 student enrolled courses")
            print("Press-6 All payments")
            print("Press-7 Assign teacher to course")
            print("Press-8 update course info")
            print("Press-9 Get course info")
            print("Press-10 Get enrolled students in a course")
            print("Press-11 Get teacher of a particular course")
            print("Press-12  student associated with the enrollment")
            print("Press-13 course associated with the enrollment")
            print("Press-14 Update teacher info")
            print("Press-15 display teacher info")
            print("Press-16 display assigned course to teacher")
            print("Press-17 Exit")
```

```
i=int(input())
if i==1:
    id=int(input("\nEnter enrollment id: "))
    s=int(input("\nEnter stundent id: "))
    c=int(input("\nEnter Course id: "))
    d=input("\nEnter enrollment date: ")
    Enrollment(enrollment_id=id,student_id=s,course_id=c,enrollment_date=d).EnrollInCou
elif i == 2:
    sid=int(input("\nEnter student id: "))
    fname=input("\nEnter New firstname: ")
    lname=input("\nEnter New lastname: ")
    dob=input("\nEnter new DOB: ")
    email=input("\nEnter new email: ")
    phoneno=input("\nEnter new phone no: ")
    r.UpdateStudentInfo(sid,fname,lname,dob,email,phoneno)
elif i == 3:
    id=int(input("\nEnter payment id: "))
    sid=int(input("\nEnter student id: "))
    a=float(input("\nEnter amount: "))
    p=input("\nEnter payment date: ")
    Payment(payment_id=id,student_id=sid,amount=a,payment_date=p).MakePayment()
elif i == 4:
    id=int(input("\nEnter studentid: "))
    r.DisplayStudentInfo(id)
```

```
elif i == 5:
    id=int(input("Enter student id: "))
    enrolled_courses=r.GetEnrolledCourses(id)
    if enrolled_courses:
        print("Courses Enrolled:")
        for course in enrolled_courses:
            print(f"Course Name: {course[0]}")
            print()
    else:
       print("No courses enrolled for this student.")
elif i == 6:
   id=int(input("\nEnter studentid: "))
   payment_history=r.GetPaymentHistory(id)
   if payment_history:
       print("Payment History:")
       for payment_record in payment_history:
           print(f"Amount: {payment_record[0]}")
           print(f"Payment Date: {payment_record[1]}")
           print()  # Empty line for separation
   else:
    print("No payment records found for this student.")
elif i == 7:
    cid=int(input("\nEnter courseid: "))
    tid=int(input("\nEnter teacher: "))
    r.AssignTeacher(cid,tid)
```

```
elif i == 8:
    cname=input("\nEnter new course name: ")
    credit=int(input("\nEnter new credtis: "))
    tid=input("\nEnter new teacher id: ")
    r.UpdateCourseInfo(cname,credit,tid)
elif i == 9:
    cname=input("\nEnter new course name: ")
    r.DisplayCourseInfo(cname)
elif i == 10:
    cname=input("\nEnter new course name: ")
    r.GetEnrollments(cname)
elif i == 11:
    cname=input("\nEnter new course name: ")
    r.GetTeacher(cname)
elif i == 12:
    id=int(input("\nEnter enrollment id: "))
    r.GetStudent(id)
elif i == 13:
    id=int(input("\nEnter enrollment id: "))
    r.GetCourse(id)
elif i == 14:
    tid=int(input("\nEnter teacher id: "))
    tfname=input("\nEnter updated first name: ")
    tlname=input("\nEnter updated last name: ")
    email=input("\nEnter new email: ")
    r.UpdateTeacherInfo(tid,email,tfname,tlname)
elif i == 15:
    tid=int(input("\nEnter teacher id: "))
    r.DisplayTeacherInfo(tid)
```

```
            elif i == 16:
                tid=int(input("\nEnter teacher id: "))
                r.GetAssignedCourses(tid)
            elif i == 17:
                print("\nThank You\n")
                break
            else:
                print('\nInvalid Choice!!\nPlease Try Again...\n')

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

Task 7: Database Connectivity

```python
class PropertyUtil:
    def getPropertyString():
        host = 'localhost'
        username = 'root'
        password = '9927559686'
        database = 'sisdb'
        return host, username, password, database
```

```python
from DBproperty import PropertyUtil
import mysql.connector as connection

class DBConnection():
    def __init__(self):
        pass

    def open(self):
        try:
            l = PropertyUtil.getPropertyString()
            self.mydb = connection.connect(host=l[0], database=l[3], username=l[1], password=l[2])
            self.c = self.mydb.cursor(buffered = True)
        except Exception as e:
            print(e)

    def close(self):
        self.c.close()
```