Assignment 2 student information system

By Pradum singh

- ***Task 1: Define Classes***
  - *Student*
    -

```
class Student:
    def __init__(self, student_id: int, first_name: str, last_name: str, date_of_birth: str, email_address: str, phone_number: str):
        self.student_id = student_id
        self.first_name = first_name
        self.last_name = last_name
        self.date_of_birth = date_of_birth
        self.email_address = email_address
        self.phone_number = phone_number
```

```
class Student:
    def __init__(self, studentID:int, firstName:str, lastName:str, dob:str, email:str, phoneNum:str):
        self.studentID = studentID
        self.firstName = firstName
        self.lastName = lastName
        self.dob = dob
        self.email = email
        self.phone = phoneNum
```

  - *Course*

```
class Course:
    def __init__(self, course_id: int, course_name: str, course_code: int, instructor_name: str):
        self.course_id = course_id
        self.course_name = course_name
        self.course_code = course_code
        self.instructor_name = instructor_name
```

    -
  - *Enrollment*

```python
from Student import Student
from Course import Course

2 usages
class Enrollment:
    def __init__(self, enrollment_id: int, student: Student, course: Course, enrollment_date: str):
        # Initialize Enrollment attributes
        self.student = student  # Contains student information
        self.course = course    # Contains course information
        self.enrollment_id = enrollment_id  # Unique enrollment identifier
        self.enrollment_date = enrollment_date  # Date of enrollment
```

- *Teacher*

```python
class Teacher:
    def __init__(self, teacher_id: int, first_name: str, last_name: str, email: str):
        # Initialize Teacher attributes
        self.teacher_id = teacher_id  # Unique teacher identifier
        self.first_name = first_name
        self.last_name = last_name
        self.email = email

#Uncomment the following code to test the Teacher class with Harry Potter details
if __name__ == '__main__':
    snape = Teacher( teacher_id: 1,  first_name: "Severus",  last_name: "Snape",  email: "snape@hogwarts.com")
```

  - Output Prediction
    - `snape = Teacher(1, "Severus", "Snape", " snape@hogwarts.com ")`

- *Payment*

```python
from Student import Student


2 usages
class Payment(Student):
    def __init__(self, payment_id: int, student: Student, amount: float, payment_date: str):
        # Inherit student details from the parent Student class
        super().__init__(student.student_id, student.first_name, student.last_name, student.date_of_birth,
                         student.email_address, student.phone_number)
        self.payment_id = payment_id  # Unique payment identifier
        self.amount = amount  # Payment amount
        self.payment_date = payment_date  # Date of payment
```

- ***Task 2: Implement Constructors & Methods***
  - +++++++

- ***Task 3: Implement Methods***
  - *Methods of Student Class*
  - *Course Class*
  - *Enrollment Class*
  - *Teacher Class*
  - *Payment Class*
  - *SIS class to manage interactions*

```python
from datetime import datetime

class HogwartsStudentPortal:
    def __init__(self, database_util):
        self.database_util = database_util

    def get_student_information(self):
        query = ("SELECT student_id, CONCAT(first_name, ' ', last_name) AS 'Name', date_of_birth, email, phone_number "
                 "FROM students")
        result = self.database_util.fetchall(query)
        return result

    def add_new_student(self):
        student_id = self.generate_unique_student_id()
        print("Fill up the student details:")
        student_info = {
            'student_id': student_id,
            'first_name': input("Enter the First Name: "),
            'last_name': input("Enter the Last Name: "),
            'date_of_birth': input("Enter the date of birth: "),
            'email': input("Enter the email ID: "),
            'phone': input("Enter the phone number:")
        }
        if not self.check_email_id(student_info['email']):
            raise Exception("Email ID already exists")
        if not self.check_phone_number(student_info['phone']):
            raise Exception("Phone number already exists")

        query = "INSERT INTO students VALUES (%s, %s, %s, %s, %s, %s)"
        values = (student_info['student_id'], student_info['first_name'], student_info['last_name'],
                  student_info['date_of_birth'], student_info['email'], student_info['phone'])
        return self.database_util.execute_query(query, values)
```

```
def enroll_student_in_course(self):
    enrollment_id = self.generate_unique_enrollment_id()
    enrollments = {
        'enrollment_id': enrollment_id,
        'student_id': input("Enter the student ID: "),
        'enrollment_date': datetime.now().strftime("%Y-%m-%d")
    }
    query = "INSERT INTO enrollments VALUES (%s, %s, %s, %s)"
    print("Choose which Course you want to enroll in:")
    print("1. Introduction to Magic")
    print("2. Potions 101")
    print("3. Spell Casting")
    print("4. Defense Against the Dark Arts")
    print("5. Magical Creatures Study")
    choice = int(input("Enter: "))
    if 1 <= choice <= 5:
        enrollments['course_id'] = f'C00{choice}'
        values = (enrollments['enrollment_id'], enrollments['student_id'], enrollments['course_id'],
                  enrollments['enrollment_date'])
        return self.database_util.execute_query(query, values)
    else:
        print("Invalid choice. Enrollment aborted.")
        return None

1 usage
def get_no_of_students(self):
    query = "SELECT COUNT(*) FROM students"
    result = self.database_util.fetch_one(query)
    return result[0]

1 usage
def get_no_of_enrollments(self):
    query = "SELECT COUNT(*) FROM enrollments"
    result = self.database_util.fetch_one(query)
    return result[0]
```

- ++
-

```
1 usage
def get_no_of_enrollments(self):
    query = "SELECT COUNT(*) FROM enrollments"
    result = self.database_util.fetch_one(query)
    return result[0]

1 usage
def generate_unique_student_id(self):
    return f'ST{self.get_no_of_students() + 1:03d}'

1 usage
def generate_unique_enrollment_id(self):
    return f'EE{self.get_no_of_enrollments() + 1:03d}'

1 usage
def check_email_id(self, email):
    query = "SELECT email FROM students"
    result = self.database_util.fetchall(query)
    return email not in result[0]

1 usage
def check_phone_number(self, phone):
    query = "SELECT phone_number FROM students"
    result = self.database_util.fetchall(query)
    return phone not in result[0]
```

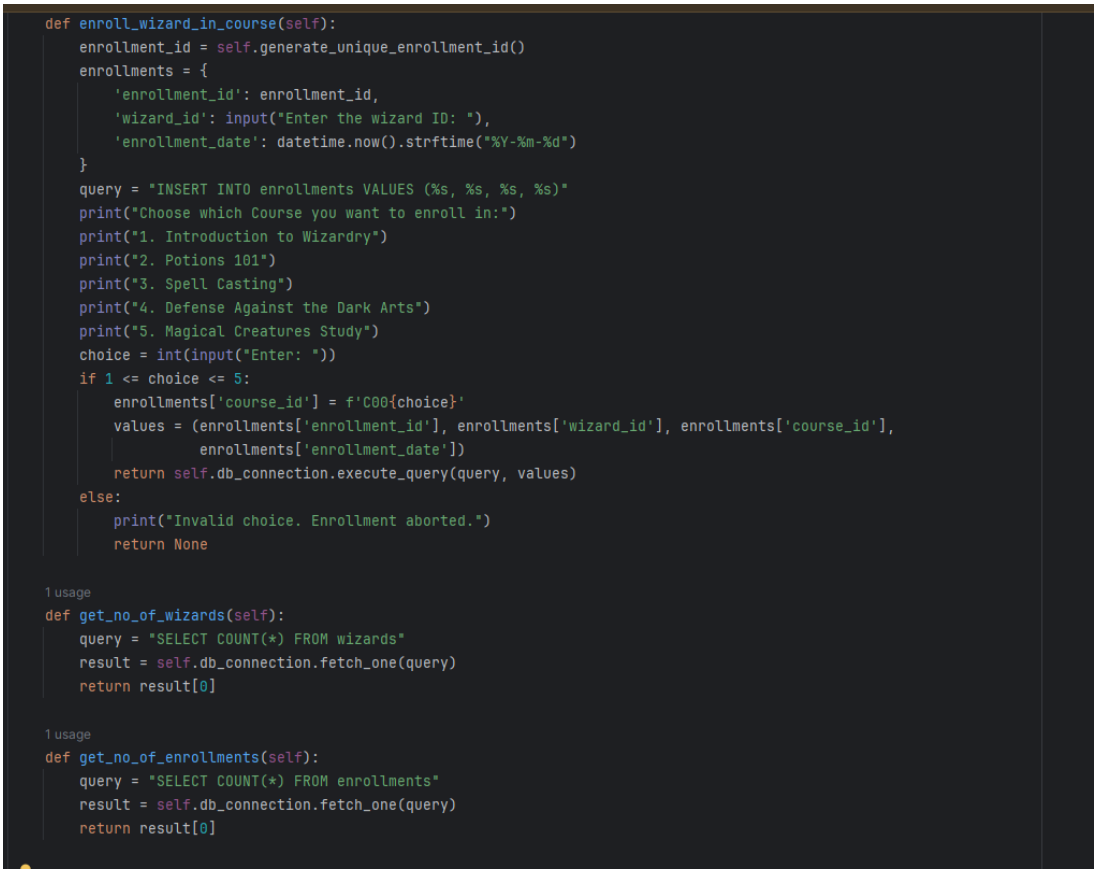
  - *Constructors*
  - *SIS class*
-
- ***Task 4: Exceptions handling and Custom Exceptions***
  - `
- ***Task 5: Collections [IT has been implemented please refer my project directory]***
- ***Task 6: Create Methods for Managing Relationships***
  - [Check directory for better view]
- ***Task 7: Database Connectivity***
  -

```python
class DBUtil:
    def __init__(self, host, user, password, port, database):
        self.connection = connect(
            host=host,
            user=user,
            password=password,
            port=port,
            database=database
        )
        self.cursor = self.connection.cursor()

    5 usages (5 dynamic)
    def executeQuery(self, query, values=None):
        try:
            self.cursor.execute(query, values)
            self.connection.commit()
        except Exception as e:
            print(f"QueryExecution Error: {e}")
            self.connection.rollback()

    9 usages (9 dynamic)
    def fetchall(self, query, values=None):
        try:
            self.cursor.execute(query, values)
            return self.cursor.fetchall()
        except Exception as e:
            print(f"FetchAll Error: {e}")
            self.connection.rollback()

    3 usages (3 dynamic)
    def fetchOne(self, query, values=None):
        try:
            self.cursor.execute(query, values)
            return self.cursor.fetchone()
        except:
            print(f"FetchOne Error!")
            self.connection.rollback()
```

- ***Task 8: Student Enrollment***
  -

```python
from datetime import datetime

class WizardryStudentPortal:
    def __init__(self, db_connection):
        self.db_connection = db_connection

    def get_wizard_information(self):
        query = ("SELECT wizard_id, CONCAT(first_name, ' ', last_name) AS 'Name', date_of_birth, owl_email, broomstick_number "
                 "FROM wizards")
        result = self.db_connection.fetchall(query)
        return result

    def register_new_wizard(self):
        wizard_id = self.generate_unique_wizard_id()
        print("Fill up the wizard details:")
        wizard_info = {
            'wizard_id': wizard_id,
            'first_name': input("Enter the First Name: "),
            'last_name': input("Enter the Last Name: "),
            'date_of_birth': input("Enter the date of birth: "),
            'owl_email': input("Enter the Owl Email: "),
            'broomstick_number': input("Enter the Broomstick Number:")
        }
        if not self.check_owl_email(wizard_info['owl_email']):
            raise Exception("Owl Email already exists")
        if not self.check_broomstick_number(wizard_info['broomstick_number']):
            raise Exception("Broomstick Number already exists")

        query = "INSERT INTO wizards VALUES (%s, %s, %s, %s, %s, %s)"
        values = (wizard_info['wizard_id'], wizard_info['first_name'], wizard_info['last_name'],
                  wizard_info['date_of_birth'], wizard_info['owl_email'], wizard_info['broomstick_number'])
        return self.db_connection.execute_query(query, values)

    def enroll_wizard_in_course(self):
        enrollment_id = self.generate_unique_enrollment_id()
        enrollments = {
            'enrollment_id': enrollment_id,
            'wizard_id': input("Enter the wizard ID: "),
            'enrollment_date': datetime.now().strftime("%Y-%m-%d")
```

- ++
  -



```
def enroll_wizard_in_course(self):
    enrollment_id = self.generate_unique_enrollment_id()
    enrollments = {
        'enrollment_id': enrollment_id,
        'wizard_id': input("Enter the wizard ID: "),
        'enrollment_date': datetime.now().strftime("%Y-%m-%d")
    }
    query = "INSERT INTO enrollments VALUES (%s, %s, %s, %s)"
    print("Choose which Course you want to enroll in:")
    print("1. Introduction to Wizardry")
    print("2. Potions 101")
    print("3. Spell Casting")
    print("4. Defense Against the Dark Arts")
    print("5. Magical Creatures Study")
    choice = int(input("Enter: "))
    if 1 <= choice <= 5:
        enrollments['course_id'] = f'C00{choice}'
        values = (enrollments['enrollment_id'], enrollments['wizard_id'], enrollments['course_id'],
                  enrollments['enrollment_date'])
        return self.db_connection.execute_query(query, values)
    else:
        print("Invalid choice. Enrollment aborted.")
        return None

1 usage
def get_no_of_wizards(self):
    query = "SELECT COUNT(*) FROM wizards"
    result = self.db_connection.fetch_one(query)
    return result[0]

1 usage
def get_no_of_enrollments(self):
    query = "SELECT COUNT(*) FROM enrollments"
    result = self.db_connection.fetch_one(query)
    return result[0]
```

```
1 usage
def get_no_of_wizards(self):
    query = "SELECT COUNT(*) FROM wizards"
    result = self.db_connection.fetch_one(query)
    return result[0]

1 usage
def get_no_of_enrollments(self):
    query = "SELECT COUNT(*) FROM enrollments"
    result = self.db_connection.fetch_one(query)
    return result[0]

1 usage
def generate_unique_wizard_id(self):
    return f'WZ{self.get_no_of_wizards() + 1:03d}'

1 usage
def generate_unique_enrollment_id(self):
    return f'EN{self.get_no_of_enrollments() + 1:03d}'

1 usage
def check_owl_email(self, owl_email):
    query = "SELECT owl_email FROM wizards"
    result = self.db_connection.fetchall(query)
    return owl_email not in result[0]

1 usage
def check_broomstick_number(self, broomstick_number):
    query = "SELECT broomstick_number FROM wizards"
    result = self.db_connection.fetchall(query)
    return broomstick_number not in result[0]
```

- ***Task 9: Teacher Assignment***
  -

```python
from datetime import datetime

class WizardryPaymentService:
    def __init__(self, db_connection):
        self.db_connection = db_connection

    def add_new_payment_record(self):
        payment_id = self.generate_unique_payment_id()
        print("Fill up the Magical Payment details:")
        payments = {
            'payment_id': payment_id,
            'wizard_id': input("Enter the Wizard ID: "),
            'amount': float(input("Enter the amount: ")),
            'payment_date': input("Enter the Payment Date in (YYYY-MM-DD) format: ")
        }
        query = "INSERT INTO magical_payments VALUES (%s, %s, %s, %s)"
        values = (payments['payment_id'], payments['wizard_id'], payments['amount'], payments['payment_date'])
        return self.db_connection.execute_query(query, values)

    def get_payment_details(self):
        wizard_id = input("Enter your Wizard ID: ")
        query = "SELECT * FROM magical_payments WHERE wizard_id = %s"
        values = (wizard_id,)
        result = self.db_connection.fetchall(query, values)
        return result

    def update_payment_records(self):
        payment_id = input("Enter your Payment ID: ")
        amount = input("Enter the amount")
        date = datetime.now().strftime("%Y-%m-%d")
        query = "UPDATE magical_payments SET amount=%s, payment_date=%s WHERE payment_id=%s"
        values = (amount, date, payment_id)
        return self.db_connection.execute_query(query, values)

    1 usage
    def get_no_of_payments(self):
        query = "SELECT COUNT(*) FROM magical_payments"
        result = self.db_connection.fetch_one(query)
        return result[0]
```

- ++
-

- ***Task 10: Payment Record***
  -

```python
from datetime import datetime

class WizardryPaymentService:
    def __init__(self, db_connection):
        self.db_connection = db_connection

    def add_new_payment_record(self):
        payment_id = self.generate_unique_payment_id()
        print("Fill up the Magical Payment details:")
        payments = {
            'payment_id': payment_id,
            'wizard_id': input("Enter the Wizard ID: "),
            'amount': float(input("Enter the amount: ")),
            'payment_date': input("Enter the Payment Date in (YYYY-MM-DD) format: ")
        }
        query = "INSERT INTO magical_payments VALUES (%s, %s, %s, %s)"
        values = (payments['payment_id'], payments['wizard_id'], payments['amount'], payments['payment_date'])
        return self.db_connection.execute_query(query, values)

    def get_payment_details(self):
        wizard_id = input("Enter your Wizard ID: ")
        query = "SELECT * FROM magical_payments WHERE wizard_id = %s"
        values = (wizard_id,)
        result = self.db_connection.fetchall(query, values)
        return result

    def update_payment_records(self):
        payment_id = input("Enter your Payment ID: ")
        amount = input("Enter the amount")
        date = datetime.now().strftime("%Y-%m-%d")
        query = "UPDATE magical_payments SET amount=%s, payment_date=%s WHERE payment_id=%s"
        values = (amount, date, payment_id)
        return self.db_connection.execute_query(query, values)
```

```python
    def update_payment_records(self):
        payment_id = input("Enter your Payment ID: ")
        amount = input("Enter the amount")
        date = datetime.now().strftime("%Y-%m-%d")
        query = "UPDATE magical_payments SET amount=%s, payment_date=%s WHERE payment_id=%s"
        values = (amount, date, payment_id)
        return self.db_connection.execute_query(query, values)

    1 usage
    def get_no_of_payments(self):
        query = "SELECT COUNT(*) FROM magical_payments"
        result = self.db_connection.fetch_one(query)
        return result[0]

    1 usage
    def generate_unique_payment_id(self):
        return f'MP{self.get_no_of_payments() + 1:03d}'
```

- ***Task 11: Enrollment Report Generation***
  -

```python
class EnchantmentReportGeneration:
    def __init__(self, magical_db_util):
        self.magical_db_util = magical_db_util

    def generate_enchantment_report(self):
        course_code = input("Enter the course code: ")
        query = "SELECT * FROM magical_courses mc JOIN magical_enrollments me ON mc.course_code = me.course_code WHERE mc.course_code = %s"
        value = (course_code,)
        return self.magical_db_util.fetchall(query, value)
```