OBJECT ORIENTED PROGRAMMING THROUGH PYTHON

REPORT OF SUPPLEMENTAL CA

School Management System

Paulina Czarnota C21365726
Due Date: 04/08/2023

INTRODUCTION

This report explains the implementation of the School Management System, which is a Python-based application developed to manage various aspects of a school, such as storing student information, computing final grades, generating student reports, and providing CRUD operations for student data. The system is designed using object-oriented programming (OOP) principles, and data persistence is ensured through the use of external files. This report will discuss the project classes, methods, a user manual, and the challenges faced during development.

PROJECT CLASSES AND METHODS

The School Management System is built using object-oriented programming (OOP) principles and employs the following classes:

1. **Student Class:**

   - Attributes: The **Student** class contains three attributes - **name**, **student_id**, and **subject**.
   - Methods:
     - **__init__(self, name, student_id, subject)**: Initializes the student attributes.
     - **__str__(self)**: Returns a string representation of the student object.
     - **compute_final_grade(self)**: Abstract method to compute the final grade (implemented in subclasses).
     - **print_info(self)**: Abstract method to print student information (implemented in subclasses).
     - **name()**: Getter for the student's name.
     - **student_id()**: Getter for the student's ID.
     - **subject()**: Getter for the student's subject.

2. **MathStudent Class (Subclass of Student):**

   - Attributes: The **MathStudent** class has attributes - **quizzes** (a list of quiz scores), **test1_score**, **test2_score**, and **final_exam_score**.
   - Methods:
     - **__init__(self, name, student_id)**: Initializes MathStudent attributes and calls the superclass constructor.
     - **add_quiz_score(self, score)**: Adds a quiz score to the list of quizzes for the Math student.

- **compute_quiz_average(self)**: Computes the quiz average for the Math student.
- **compute_final_grade(self)**: Computes the final grade for the Math student based on the grading criteria.
- **print_info(self)**: Generates a detailed report of Math student information.

3. **HistoryStudent Class (Subclass of Student):**

- Attributes: The **HistoryStudent** class has attributes - **attendance_score**, **project_score**, **exam1_score**, and **exam2_score**.
- Methods:
  - **__init__(self, name, student_id)**: Initializes HistoryStudent attributes and calls the superclass constructor.
  - **compute_final_grade(self)**: Computes the final grade for the History student based on the grading criteria.
  - **print_info(self)**: Generates a detailed report of History student information.

4. **EnglishStudent Class (Subclass of Student):**

- Attributes: The **EnglishStudent** class has attributes - **attendance_score**, **final_exam_score**, **quiz1_score**, and **quiz2_score**.
- Methods:
  - **__init__(self, name, student_id)**: Initializes EnglishStudent attributes and calls the superclass constructor.
  - **compute_final_grade(self)**: Computes the final grade for the English student based on the grading criteria.
  - **print_info(self)**: Generates a detailed report of English student information.

5. **School Class:**

- Attributes: The **School** class has an attribute called **students**, which is a list to store student objects.
- Methods:
  - **__init__(self)**: Initializes the School object with an empty list of students.
  - **add_student(self, student)**: Adds a student to the school.

- **remove_student(self, student_id)**: Removes a student from the school based on the student ID.
- **find_student_by_id(self, student_id)**: Finds a student by student ID.
- **print_all_students_info(self)**: Prints information of all students in the school.
- **read_student_data(self, subject)**: Reads student data from a file for a given subject.
- **write_student_data(self, subject)**: Writes student data to a file for a given subject.
- **generate_student_reports(self)**: Generates and prints student reports for all students in the school.

6. login function:

- This function simulates the login process for user authentication. In this project, it always returns True.

7. update_student_data function:

- This function updates student data based on the student type (Math, History, or English). It prompts the user to provide updated data for the specific subject.

8. main function:

- The main function is the entry point of the program.
- It checks for successful login using the login function.
- It creates a School object to manage students and reads existing student data from files for Math, History, and English subjects.
- It presents a menu-based interface to perform various tasks related to managing students (e.g., view all students, add a new student, remove a student, update student data, generate student reports, and exit the program).
- The user's input determines which task is executed.

USER MANUAL

To use the School Management System, follow these steps:

1. **Login**: At the start, the system will prompt for a username and password. The default username is "admin," and the password is "password." Enter the credentials to access the system.

2. **Main Menu**: After successful login, the user will be presented with a menu-based interface with numbered options. Choose an option (1-6) to perform specific tasks:
    - **Option 1**: View all students and their basic information (subject, name, and student ID).
    - **Option 2**: Add a new student to the system. Select the subject (Math, History, or English) by entering 1, 2 or 3, then provide the student's name and student ID.
    - **Option 3**: Remove a student from the system by entering the student's ID.
    - **Option 4**: Update student data, such as quiz scores, test scores, attendance, and final exam scores. To update student data, provide the student's ID and follow the prompts.
    - **Option 5**: Generate and print comprehensive student reports, including final grades and assessment component scores.
    - **Option 6**: Exit the program.

DIFFICULTIES AND CHALLENGING PARTS

During the development of the School Management System, I encountered several difficulties and challenges that required diligent problem-solving. The first major obstacle was implementing file data persistence, as issues with data formatting and validation arose when reading and writing student data to external files. Robust error-checking mechanisms were essential to prevent data corruption from incomplete or invalid values. Secondly, crafting a user-friendly Command Line Interface (CLI) demanded careful planning and organization to create an intuitive menu-based system and handle user inputs gracefully. Validating inputs and maintaining a concise interface were also crucial aspects. Another significant challenge was designing the logic to compute final grades for various student types based on different assessment components, such as quizzes, tests, attendance, and projects. Balancing grading criteria and generating well-formatted student reports proved to be intricate tasks. Despite these challenges, I persevered through testing and iterative improvements, applying the principles of object-oriented programming and breaking down the problems into manageable pieces. Ultimately, I successfully delivered a functional and user-friendly School Management System.

CODE DEMO

A Code Demo is available in MP4 format, showcasing the functionality and features of the School Management System. It has been submitted as a separate file on Brightspace.

CONCLUSION

In conclusion, the School Management System project showcases the practical use of object-oriented programming in Python, resulting in an efficient and flexible system. With well-designed classes and methods, student data can be managed seamlessly, and detailed reports can be generated. The user manual and code demo further illustrate the system's capabilities. Despite challenges, the project's success underscores the benefits of OOP and systematic approach in software development.