COURSE WORK: PROGRAMMING FOR DATA SCIENCE
2024.2 BATCH

MSc Data Science: Coventry University UK

Author: B M J N Balasuriya
Index: COMScDS242P-009

Report: School Management System Using Object-Oriented Programming (OOP)

Introduction:
The School Management System is designed to handle various roles within a school, including students, teachers, and staff. The system leverages Object-Oriented Programming (OOP) principles such as inheritance, encapsulation, polymorphism, and method overriding to manage responsibilities, track information, and perform role-specific actions efficiently.

Git repository: https://github.com/j2damax/programming-data-science-cw

- Created a public git repository with readme file
- Readme file created with project overview and setup instructions.
- Created .gitignore file to avoid unnecessary files in the git repository.

Key Features and Implementation

a). Inheritance in School Classes:

Objective:
- Reuse common attributes and methods across different roles (students, teachers, staff).

Implementation:
- A base class Person was created with common attributes like name, age, address and ssn.
- Subclasses Student, Teacher, and Staff inherit from Person and add specialized attributes:
- Student: student_id
- Teacher: teacher_id
- Staff: staff_id

Demonstration:
- Instances of Student, Teacher, and Staff were created, and their common and specialized attributes were displayed.
- In the main.py class demonstrate_inheritance() demonstrate this feature in the code. When running the code choose option a) Demonstrate Inheritance.

b). Assigning Grades Method for Students:
Objective:
- Allow students to receive grades for different subjects and calculate their average grade.

Implementation:
- The assign_grades() method in the Student class accepts a dictionary of subjects and grades, stores them, and calculates the average grade.
- Graded validated as a dictionary input

Demonstration:
- A Student was assigned grades for Maths, Science, and English, and their average grade was calculated and displayed.
- In the main.py class demonstrate_assigning_grades() demonstrate this feature in the code. When running the code choose option b) Demonstrate Assigning Grades.

c). Encapsulation for Sensitive Information:

Objective:
- Protect sensitive data like Social Security Numbers (SSN) using getter and setter methods.

Implementation:
- The ssn attribute in the Person class was encapsulated using private access (_ssn).
- Getter (get_ssn()) and setter (set_ssn()) methods were implemented to securely access and modify the ssn.

Demonstration:
- The ssn of a Student and Teacher was accessed and updated using the getter and setter methods.
- In the main.py class demonstrate_encapsulation() demonstrate this feature in the code. When running the code choose option c) Demonstrate Encapsulation.

d). Class-Specific Responsibilities with Method Overriding:

Objective:
- Handle different roles (students, teachers, staff) uniformly while respecting their unique responsibilities.

Implementation:
- A role_duties() method was defined in the Person class and overridden in each subclass to provide role-specific behavior.
- Polymorphism was demonstrated by calling role_duties() on a list of Person objects, dynamically resolving the appropriate method for each role.

Demonstration:
- The system displayed the specific responsibilities of a Student, Teacher, and Staff using a unified interface.
- In the main.py class demonstrate_role_duties() demonstrate this feature in the code. When running the code choose option d) Demonstrate Role-Specific Responsibilities.

e). Specialized Teacher Class:

Objective:
- Manage teacher-specific responsibilities like handling subjects and class schedules.

Implementation:
- The Teacher class includes attributes like subject and class_schedule.
- The schedule_classes() method allows teachers to assign and update their class schedules.

Demonstration:
- A Teacher specializing in Mathematics was assigned a schedule, which was updated and displayed.
- In the main.py class demonstrate_teacher_class() demonstrate this feature in the code. When running the code choose option e) Demonstrate the Specialized Teacher Class.

f). Attendance Tracking for Students:

Objective:
- Track whether a student attended or missed a class and calculate their overall attendance.

Implementation:
- The attendance() method in the Student class records attendance for each class.
- The get_attendance_summary() method calculates and displays the total classes attended, total classes, and attendance percentage.

Demonstration:
- A Student attended Maths and English but missed Science. Their overall attendance summary was displayed.
- In the main.py class demonstrate_attendance_tracking() demonstrate this feature in the code. When running the code choose option f) Demonstrate Attendance Tracking for Students.

g). Staff Salary Management:

Objective:
- Calculate and manage the salary of staff members based on their base salary and years of service.

Implementation:
- The calculate_salary() method in the Staff class calculates the salary by adding a yearly bonus to the base salary.
- The get_salary() method retrieves the calculated salary.

Demonstration:
- A Staff member with a base salary of Rs. 35,000 and 5 years of service was assigned a yearly bonus of Rs. 2,000. Their calculated salary was displayed.
- In the main.py class demonstrate_staff_salary_management() demonstrate this feature in the code. When running the code choose option g) Demonstrate Staff Salary Management.

h. Polymorphism in School System:
Objective:
- Handle different roles (students, teachers, staff) uniformly while respecting their unique responsibilities.

Implementation:
- A role_duties() method was defined in the Person class and overridden in each subclass to provide role-specific behavior.
- Polymorphism was demonstrated by calling role_duties() on a list of Person objects, dynamically resolving the appropriate method for each role.

Demonstration:
- The system displayed the specific responsibilities of a Student, Teacher, and Staff using a unified interface.
- In the main.py class demonstrate_polymorphism() demonstrate this feature in the code. When running the code choose option h) Demonstrate Polymorphism.