OOP in C++ Final Project

Project's assumptions

I have developed an application that utilizes object-oriented programming to allow a teacher to:

- Assign 1, 2, or 3 grades to students.
- Calculate the average grade based on the number of available grades.
- Identify the student with the highest average grade, but only among those who have 3 grades.

Key Considerations:

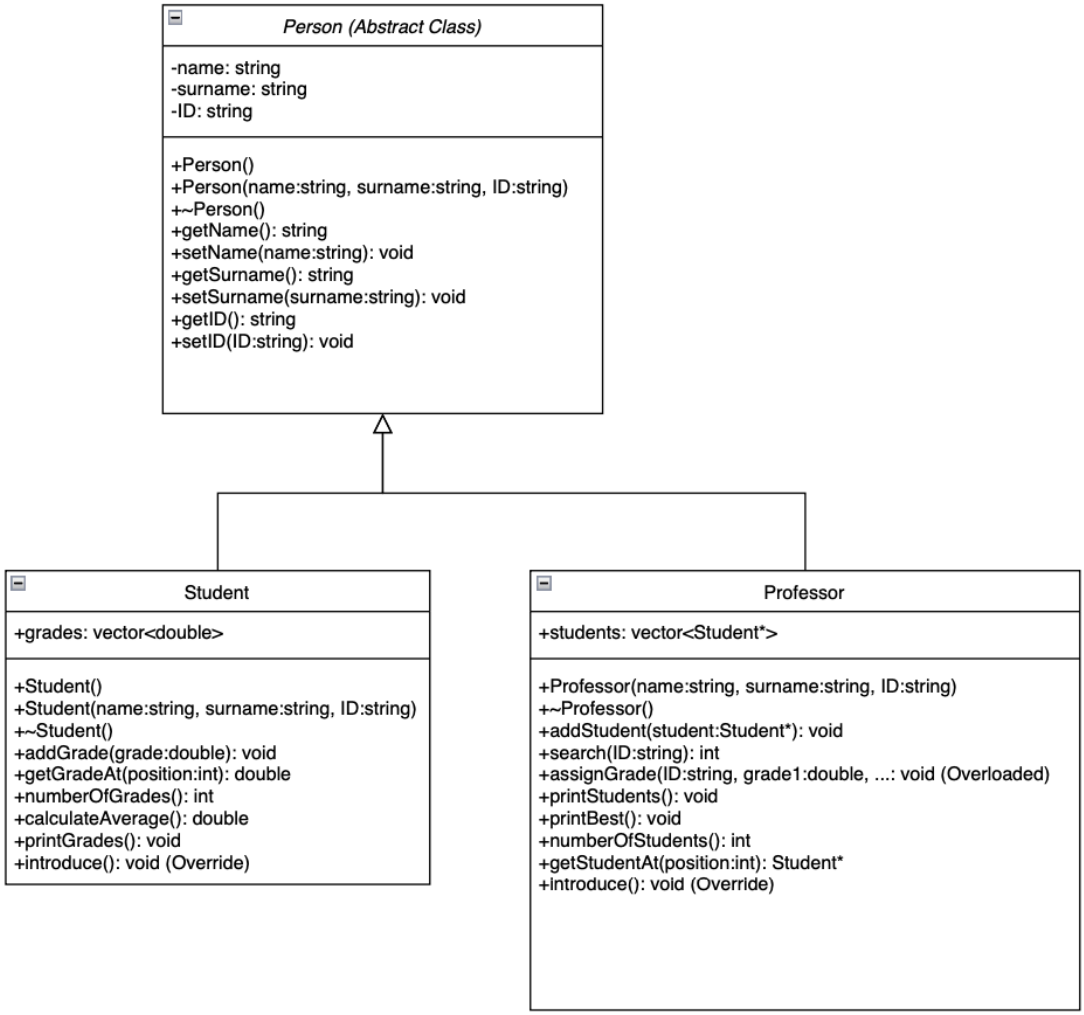
1. **Inheritance Structure**: both Teacher and Student inherit from a common base class, sharing common members (attributes and methods).
2. **Base Class Constructor**: the base class has an overloaded constructor that allows the creation of both Teacher and Student objects. This constructor accepts parameters for name, surname, and ID.
3. **Data Entry**: the application does not include any menu for data entry. All data is created at compile time.
4. **Inheritance and Polymorphism**: Through inheritance, both the Teacher and Student classes derive common attributes and functionalities from a base class, while also having unique methods. Polymorphism is used in method overloading for assigning grades.
5. **Data Management**: The application efficiently manages student data, allowing the teacher to add students and assign varying numbers of grades, reflecting a real-world scenario where not all students might have the same number of assessments.
6. **Calculation and Display of Results**: The program can calculate the average grade for each student and determine the student with the highest average, but it specifically focuses on those with a complete set of three grades, showcasing a selective data processing approach.
7. **Encapsulation**: Data integrity is maintained by encapsulating the details within each class, exposing only necessary functionalities to the user (the teacher in this context).

Class Details:

1. **Shared Data**: Students and teachers share data such as name, surname, and ID. These objects are created with these attributes, utilizing an overloaded constructor.
2. **Teacher Capabilities**:
   a. Add a new student to their student list.
   b. Assign 1, 2, or 3 grades to a student (method overloading).
   c. Obtain the average grade of a student (average of 1, 2, or 3 grades).
   d. Print a list of students, showing their data and average grade.
   e. Print the student with the best average grade (among those with 3 grades).
3. **Student Capabilities**:
   a. Add a new grade to their list of grades.
   b. Retrieve each of their grades from their list.
   c. Return the number of grades they have (size of the list).

The program is designed to simulate a grading system where a teacher can manage and evaluate student performance. It demonstrates the principles of object-oriented programming, including inheritance, polymorphism, and encapsulation.In summary, it will display on the screen a list of the teacher's students and highlight the student with the best average grade among those who have a complete set of three grades. This practice is an effective demonstration of using object-oriented programming concepts to create a practical and interactive grading system.

Block diagram



Description of operation

The program you're describing is a simple educational management system implemented in C++ using object-oriented programming principles. It is designed to manage students and professors, allowing for the assignment and management of grades. Below is a detailed description of the operations performed by the program:

Initialization:
1. Main Entry (main.cpp):
   - The `main()` function starts by calling the `Grading()` function, which encapsulates the main logic of the program.

Setup:
2. Creating Professors and Students:
   - Instances of `Professor` and `Student` objects are created using overloaded constructors that accept name, surname, and ID as parameters.
   - Professors are created as `Person` pointers and then cast to `Professor` when needed, demonstrating polymorphism.
   - Students are first created as `Person` pointers for the demonstration of polymorphism and later directly as `Student` objects.

Operations:
3. Adding Students to Professors:
   - Students are added to a professor's list using the `addStudent()` method. This builds the relationship where a professor has multiple students.

4. Assigning Grades:
   - Grades are assigned to students using overloaded `assignGrade()` methods within the `Professor` class. These methods allow for the assignment of 1 to 5 grades at once.

5. Calculating and Displaying Averages:
   - Each student's average grade is calculated using the `calculateAverage()` method in the `Student` class. This method sums all grades and divides by the number of grades to get the average.

6. Identifying the Best Student:
   - The `Professor` class has a method `printBest()`, which iterates through the list of students who have exactly three grades and identifies the student with the highest average grade.

Interaction:
7. Printing Student and Professor Data:
   - The program prints the list of students for each professor, along with their ID, grades, and average using the `printStudents()` method.
   - It also prints the student with the best average using the `printBest()` method.

8. Searching by ID:
   - The user is prompted to enter an ID, which the program uses to search through the professors' lists. If a matching ID is found, it prints the corresponding professor or student information, along with the student's grades and average.

Cleanup:
9. Memory Management:
   - Once the operations are complete, the program deallocates the memory used for professors and students using `delete` to prevent memory leaks.

Abstract Base Class (Person):
10. Person Class:
    - This is the base class for both `Student` and `Professor`, containing common attributes like name, surname, and ID, along with their getters and setters.
    - It includes a pure virtual function `introduce()`, making it an abstract class. This means it cannot be instantiated and requires derived classes to provide an implementation for the `introduce()` method.

Derived Classes (Student and Professor):
11. Student Class:
    - Inherits from `Person` and represents a student with additional attributes like a vector of grades.
    - It overrides the `introduce()` method from `Person` and provides additional methods for managing grades.

12. Professor Class:
    - Also inherits from `Person` and represents a professor with additional attributes like a list of pointers to `Student` objects.
    - It overrides the `introduce()` method from `Person` and provides methods for managing students and grades.

Program Output:
- The program outputs the list of students and their grades, the best student for each professor, and allows searching for individuals by ID, printing detailed information about the found person.

This description covers the basic operation of the program from initialization to cleanup. The program demonstrates core object-oriented programming concepts such as inheritance, polymorphism, encapsulation, and abstraction.

Link to sourcecode in GIT repository (github, gitlab, sourcetree)

https://github.com/ppaner00/OOP-final-project.git