Person Class Analysis

This class is the basis for our Student and Teacher classes. It privately declares the name and address variables following the best practices for encapsulating the variables as far as declaration goes. However, it does not follow best practices for keeping that data encapsulated due to the getters and setters for the name and address. The getName, getAddress, and setAddress methods make the fact that the name and address variables are private negligible.

An example of a replacement for setAddress would be a method called moveHome. It is more behavior driven rather than data driven and can more easily be understood to handle more information such as validations or other information that may change should more data be added to the class.

This class also provides the base implementation of toString that returns a Person's name and address in a particular format. I personally find the name to be not useful. Instead, I would name it something like getPersonalInfo to make it more clear about what the function does.

Overall, this class could be improved in terms of following the best encapsulation guidelines.

Student Class Analysis

This class inherits the Person class, which therefore means it has the class variables and methods of the Person class at its disposal. However, I do think that the addCourseGrade class is a better implementation of a behavior driven setter than the setAddress function of the parent class Person. However, it lacks the validations that it should consider, such as if you are attempting to add a course when you have already had the maximum number of classes.

This class overrides the toString method of the parent class to prepend the string "Student: " before calling the Person class' toString method. This is an example of polymorphism as the Student class is creating a new implementation of the same named function of its parent class.

Overall, this class does a good job of encapsulating its relevant data of courses and grades, though it could be improved with preventing particular errors from occurring throughout some of its methods.

Teacher Class Analysis

This class also inherits the Person class. Similar to the Student class, I think the "setters" for this class are better named and implemented via the addCourse and removeCourse which have some validations for making sure the course isn't already added or removed respectively. However, it falls into the same pitfall of not respecting some of the other limitations set forth by the class like making sure that you cannot exceed the maximum number of courses.

This class also overrides the toString method of the parent class to prepend the string “Teacher: “ before calling the Person class’ toString method.

Overall, this class does a good job of encapsulating the relevant data of courses, though it could be improved with preventing particular errors from occurring throughout some of its methods.

Final Thoughts

My final thoughts on the classes here is that it provides some good examples of dos and don’ts for proper Modularity and Encapsulation.

All three classes show modularity by keeping the class-specific functionality in its own area. The Person class handles the base functionality of the other two classes well, though the naming convention for the toString method and setter method for addresses could be improved. The Student class is modular due to it separating out the Student-specific data. The same can be said for the Teacher class and the Teacher-specific data.

While the classes all have good encapsulation up front by declaring their personal variables as private, the getter and setter methods utilized provide little validation for data and can quickly be seen as showing errors in particular cases. There could be more work done in these areas to improve such behavior.