Engineering of Software Subsystems SWEN-262

Design Principles Activity

Instructions

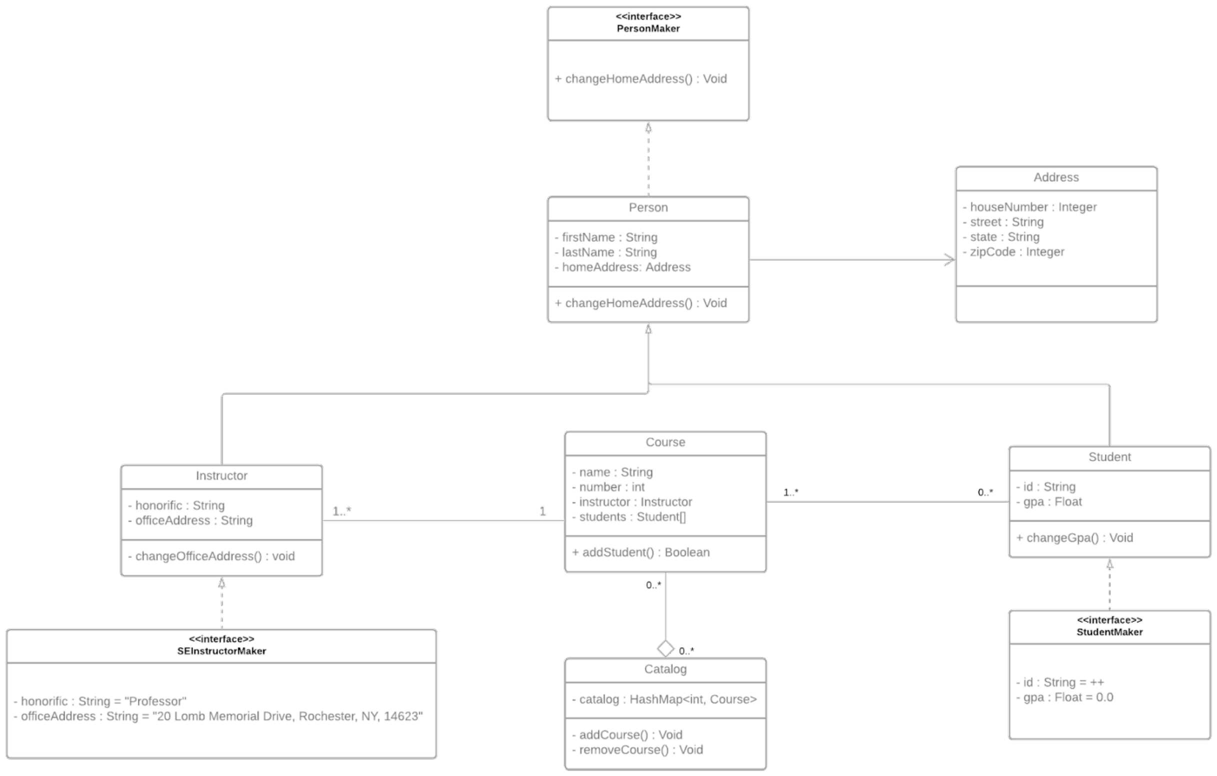
At this point, you should have completed a partial design including a UML Class Diagram and a Sequence Diagram. Consider your partial design and discuss its strengths and weaknesses in terms of at least ***four*** of the following design principles:

1. SOLID
   a. Single Responsibility Principle[1]
   b. Open/Closed Principle
   c. Liskov Substitution Principle
   d. Dependency Inversion Principle
2. GRASP
   a. Controller
   b. Information Expert
   c. High Cohesion[1]
   d. Low Coupling
   e. Polymorphism
   f. Pure Fabrication
3. Law of Demeter

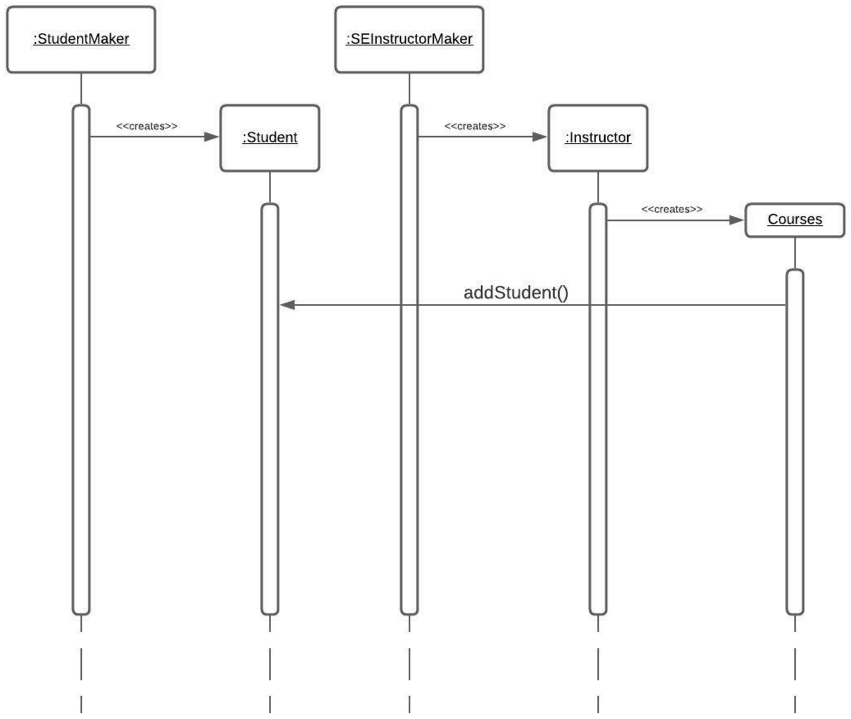
Your discussion should include one paragraph per principle that you discuss, including a detailed explanation of the principle in your own words, and a detailed analysis of how your design adheres to (or does not) including detailed examples. If you feel that you cannot write a full paragraph about a specific principle, choose a different one.

[1] To avoid redundancy, you may only choose one of these principles.

UML Review Activity
Daniel Chung



Sequence Diagram
Daniel Chung



Analysis of Your Design

1. Single responsibility: Single responsibility is when classes only have responsibility only over a single part of the program. My design does pretty well on single responsibility, because most classes only interact with their own fields, exception for courses because it takes a list of students, otherwise all the actions that classes have to take are inside their own classes.
2. Polymorphism: is when a single class can represent multiple classes. My design does that well when it's creating a class of Person and then having other classes like Instructor, and Student and extending the person class.
3. Law of Demeter: It's a principle that makes sure that classes only talk to other classes that are close instead of making a long chain of method calls. My code follows the Law of Demeter, because there's only interaction between the classes itself and at most one neighbor class. Notable interactions between my classes can be found between person and address, Course with student and instructor and Catalog with course.
4. Open-Closed Principle: Is being able to add new code when adding features instead of modifying the old one. My code does this poorly, because it lets to room for modification instead if you want to change the behavior of the code, you would have to directly modify the old code which breaks the concept of this design principle.