Assignment 7: Applying SOLID.

72 pts

Please do this independently from your classmates.

Due by end-of-day, Sunday, February 11, 2018 via a direct Slack message on the class Slack channel.

Rather than a hand-written submission, I would prefer that you edit this document directly, or attach code or UML documents separately.

CS 3398, Spring 2018.

**Context**:

We’ve recently discussed [Liskov Substitution Principle](http://www.oodesign.com/liskov-s-substitution-principle.html) and the [Interface Segregation Principle](http://www.oodesign.com/interface-segregation-principle.html). We have also discussed the [Open Closed](http://www.oodesign.com/open-close-principle.html), [Single Responsibility](http://www.oodesign.com/single-responsibility-principle.html) and [Dependency Inversion](http://www.oodesign.com/dependency-inversion-principle.html) principles. , This assignment asks you to critique and irmprove code snippets using these principles.

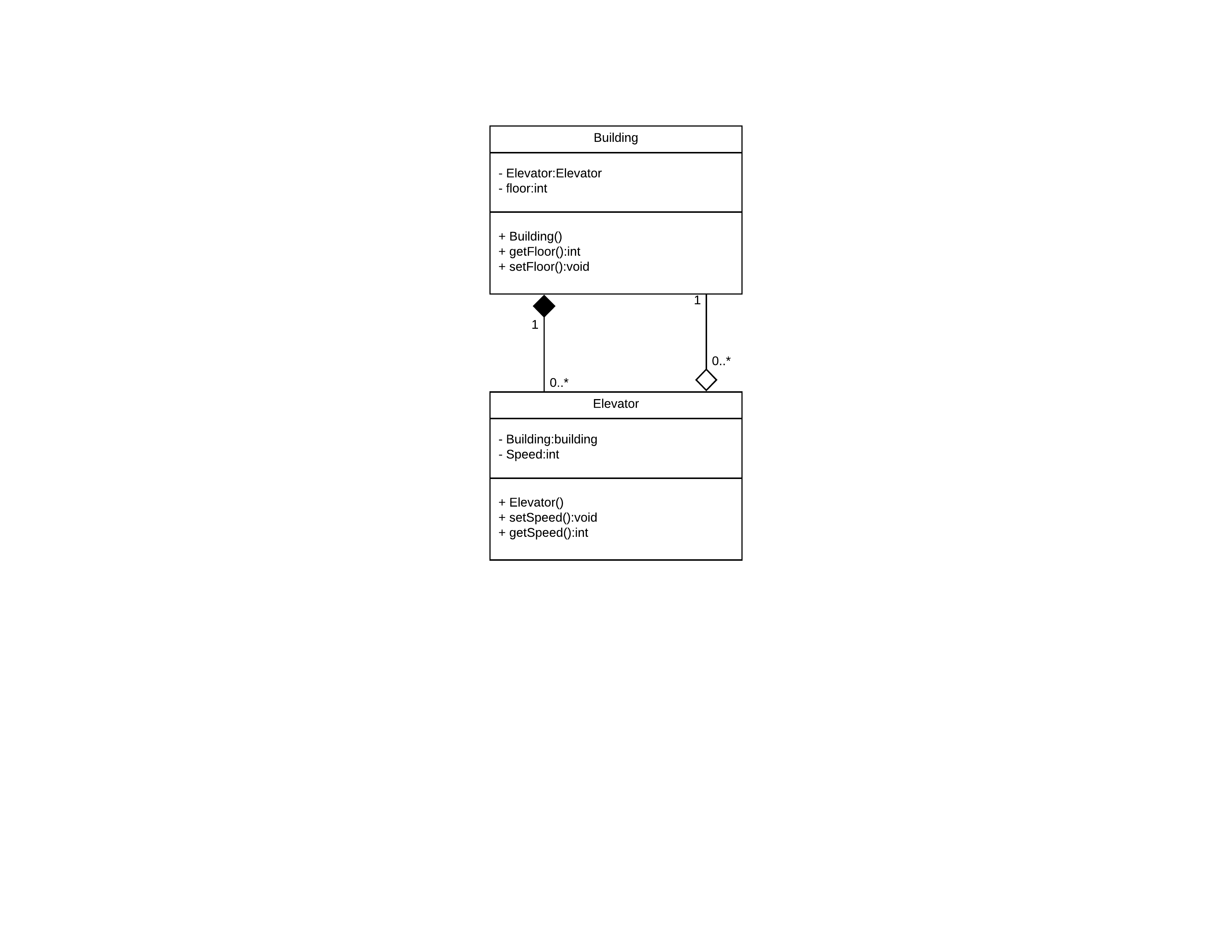
**Question 1:**  Consider the following production (shipped) code:

****

Suppose a multi storied *Building* has an *Elevator* (In UML terms, the *Building* HAS *Elevator*).

Also suppose that while constructing the *Elevator* object, we need to give it the information about the *Building* object to access some functionalities of the Building class.

In this case, the speed of the Elevator depends on the number of floors of the Building. Therefore, when constructing the Elevator object, it must access functionalities of the Building object. (In UML terms the Elevator HAS Building).

1. Draw the UML for this code (6 pts)
2. Does the code violate (explain your answers) (30 pts):
   1. **S**ingle Responsibility Principle?

No, the building’s functions are only responsible for building related needs (i.e. setting and getting the floor count). The Elevator’s functions are only responsible for elevator needs (i.e. getting and setting the speed).

* 1. **O**pen Close Principle?

Yes, If you wanted to add some more functionality in the Elevator class you would be also modifying the Building class as the building class creates an Elevator Object within its’ class and must be recompiled in order to work.

* 1. **L**iskov Substitution Principle?

No, there are no extensions in this code for Liskov Substitution to be involved.

* 1. **I**nterface Segregation Principle?

No, there are no interfaces to worry about.

* 1. **D**ependency Inversion Principle?

Yes, Building has an Elevator instance, Elevator also has Building instance in the code. Both depend on each other and have a circular dependency.

1. There is a [Circular Dependency](https://www.codeproject.com/Articles/616344/What-is-Circular-dependency-and-how-do-we-resolve) in this code. Using the concept of interfaces, change the code so that the circular dependency is gone, but the desired functionality remains (30 pts).
2. Explain your answer to (3). (6 pts)

By creating an interface for Building to implement we can strip all the functions and other useless variables of the building class for the elevator class to use. By then passing in the constructor of Elevator (in Building class) with “this”, you pass in the interfaced class with only the “getFloors()” function. The elevator class only needed the floor count and not the rest of the functions and now does not need to create its own building class and thus the circular dependency is removed.