CSCI 2300, Fall 2018

SOLID Design Principles Exercises

We have discussed five major object-oriented design principles:

S – Single Responsibility Principle (SRP)

O – Open/Closed Principle (OCP)

L – Liskov Substitution Principle (LSP)

I – Interface Segregation Principle (ISP)

D – Dependency Inversion Principle (DIP)

In this exercise, we will put theory into practice. In your git repositories, you will find a SOLID directory. This directory, contains two sub-directories: DIP and ISP. This is the code we will be using today.

Form groups with the people that are sitting in your row, and discuss your answers with your group. Review the code in the appropriate directory and answer the questions on this worksheet (one worksheet per student). Put your name at the top of the worksheet.

ISP – Interface Segregation Property

1. What is the purpose of `timeOutCallback()` method of the `Door` interface?

2. What is the purpose of `proximityCallback()` method of the `Door` interface?

3. What method(s) of the `Door` interface is `TimedDoor` class forced to implement, even though it doesn't need it?

4. What method(s) of the `Door` interface is `SensingDoor` class forced to implement, even though it doesn't need it?

5. How does this design violate Interface Segregation Property?

6. How would you change the design so that ISP is not violated? Draw out a diagram on the back of this page. Make the necessary code changes to implement the new design in your individual git repository and push your changes.

DIP – Dependency Inversion Property

1. There are two classes defined here. What is the dependency between them (which class depends on the other)?

2. If we wanted to add a `Fan` class (that can be turned on and off) to this design and a `Fan` also needed a button? One way to handle this is to add a `FanButton` class that allows the `Fan` to be turned on and off. What would the class diagram look like in this case (including `Lamp`, `Button`, `Fan`, and `FanButton`)?

3. What is the common behavior between `Lamp` and `Fan`?

4. We can define an `Equipment` interface for the common behavior of `Lamp` and `Fan` classes and have `Lamp` and `Fan` classes implement that interface. What would the class diagram look like in this case?

5. We can now modify the `Button` class to have a reference to an instance of `Equipment` object. Draw the class diagram after this modification (on the back of this page).

6. Make the necessary code changes in the DIP directory to implement the final re-design.