The University of the West Indies, St. Augustine
COMP 3607 Object Oriented Programming II
2020/2021 Semester 1
Lab Tutorial #3

This tutorial focuses on the SOLID design principles.

Learning Objectives:

- Reinforce your knowledge on each of the SOLID design principles
- Discuss examples of SOLID design principles in code snippets
- Refactor existing code to conform to one or more SOLID design principles

Section A: SOLID Knowledge

1. List the five SOLID design principles.
2. What are the key concepts of each of the SOLID design principles?
3. What are the consequences of violating each of the SOLID design principles?
4. Consider class A with the following operations:
   - Open a database connection
   - Fetch data from database
   - Write the data in an external file

(a) What is the issue with this class? Has too many responsibilities

Suppose any of the following changes happens in future.

- New database
- Adopt ORM to manage queries on database
- Change in the output structure

(b) What happens to class A in each case? will have to be modified many times

(c) What are the implications (if any) of the first change? code may have to be re-written for other operations

(d) Which SOLID principle is being violated here? SRP

(e) What is the solution to this problem according to the principle you've identified in (d) above?
make 3 classes each observing SRP

Section B: Code Examples

5. Consider an Animal parent class:

```
public class Animal {
    public void makeNoise() {
        System.out.println("I am making noise");
    }
}
```

Question 1:
1. Single Responsibility Principle
2. Open/ Closed Principle
3. Liskov Substitution Principle
4. Interface Segregation Principle
5. Dependency Inversion

Question 2:
SRP: Each class only does one thing. Makes your software easier to implement and prevents unexpected side-effects of future changes. Also makes your software easier to understand.
O/CP: Write your code so that you will be able to add new functionality without changing the existing code.
LSP: Defines that objects of a superclass shall be replaceable with objects of its subclasses without breaking the application.
ISP: Reduce the side effects and frequency of required changes by splitting the software into multiple, independent parts.
DIP: High-level modules, which provide complex logic, should be easily reusable and unaffected by changes in low-level modules, which provide utility features.

Question 3:
SRP: Code is harder to understand & debug.
O/CP: Changing one class requires you to change depending classes can become tedious.
LSP: Breaks the code.
ISP: Alot of useless methods in classes that implements it.
DIP: High-level modules are affected by low-level modules.

(a) Write code for two subclasses Dog and Cat so that we are able to replace an Animal object with the Dog or Cat without causing an error. A Dog and a Cat should make the appropriate noise for its behaviour. see related java files

(b) What happens when the following class is introduced? it crashes the application

```
class DeafDog extends Animal {
    @Override
    public void makeNoise() {
        throw new RuntimeException("I can't make noise");
    }
}
```

(c) Which principle is violated here? LSP

(d) What is the solution to this problem according to the principle you've identified in (e) above? see related java files

6. In Android, there are multiple click listeners such as *OnClickListener* and *OnLongClickListener*. The *OnClickListener* specifies the method for a callback to be invoked when a view is clicked. The*OnLongClickListener* specifies the method for a callback to be invoked when a view has been clicked and held.

(a) What is the problem with having the following interface to specify the behaviour for these two listeners? it forces classes to implement both methods

```
public interface MyOnClickListener {
    void onClick(View v);
    boolean onLongClick(View v);
}
```

(b) Which principle is violated here? ISP

(c) Why does this principle suggest using two separate interfaces? So we dont have to implement both methods

Section C: Simple Refactoring

7. Refactor the code examples in section B to adhere to the various principles see related java files

Section D: Additional Exercise

Suppose we have a system that handles authentication through external services such as Google, GitHub, etc. We would have a class for each service: *GoogleAuthenticationService*, *GitHubAuthenticationService*, etc. Now, let's say that some place in our system, we need to authenticate our user. To do that, as mentioned, we have several services available. Using code examples, explain two ways of handling this problem and discuss the strengths and weakness of each option citing the appropriate SOLID design principle(s) in play. see related java files