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

Content sourced from: See https://medium.com/mindorks/solid-principles-explained-with-examples-79d1ce114ace

Section A:
Question 1:

- **S** - Single Responsibility Principle (known as SRP)
- **O** - Open/Closed Principle
- **L** - Liskov's Substitution Principle
- **I** - Interface Segregation Principle
- **D** - Dependency Inversion Principle

Questions 2-3:
https://stackify.com/solid-design-principles/
https://stackify.com/solid-design-open-closed-principle/
https://stackify.com/solid-design-liskov-substitution-principle/
https://stackify.com/interface-segregation-principle/
https://stackify.com/dependency-inversion-principle/

Question 4:
(a) It that it handles lot of operations.
(b) So in all the cases the above class would be changed.
(c) This might affect the implementation of the other two operations as well.
(d) Single Responsibility Principle
(e) ideally according to SRP there should be three classes each having the single responsibility.

Question 5 (a)

```
public class Dog extends Animal {
    @Override
    public void makeNoise() {
        System.out.println("bow wow");
    }
}

public class Cat extends Animal {
    @Override
    public void makeNoise() {
        System.out.println("meow meow");
    }
}
```

(b) Our app will crash.
(c) Liskov substitution principle
(d) Fix: introduce interfaces.

Question 6

(a) If we have this interface then it will force clients to implement onLongClick even if they don't even care about long press. Which will lead to overhead of unused methods.
(b) Interface segregation
(c) two separate interfaces helps in removing the unused methods. If any client want both behaviours then they can implement both interfaces.

Question 7 (partial answers)

Refactoring question 6:

```
/**
 * Interface definition for a callback to be invoked when a view has
been clicked and held.
 */
public interface OnLongClickListener {
    /**
     * Called when a view has been clicked and held.
     *
     * @param v The view that was clicked and held.
     *
     * @return true if the callback consumed the long click, false
otherwise.
     */
    boolean onLongClick(View v);
}
```

```
/**
 * Interface definition for a callback to be invoked when a view is
clicked.
 */
public interface OnClickListener {
    /**
     * Called when a view has been clicked.
     *
     * @param v The view that was clicked.
     */
    void onClick(View v);
}
```

Section D

The first possibility is a dirty solution that will potentially introduce technical debt in the future; in case a new authentication service is to be integrated to the system, we will need to change the code, which as a result violates the *OCP*.

The second possibility is much cleaner, it allows for future addition of services, and changes can be done to each service without changing the integration logic. By defining a *AuthenticationService* interface and implementing it in each service, we would then be able to use Dependency Injection in our authentication logic and have our authentication method signature look something like this: *authenticate(AuthenticationService authenticationService)*. Then, we could authenticate by a specific service like this: *authenticate(new GoogleAuthenticationService)*. This helps us generalize the authentication logic without having to integrate each service separately.