**The University of the West Indies, St. Augustine**
**COMP 3607 Object Oriented Programming II**
**2019/2020 Semester 1**
**Lab Tutorial #4 Solutions**

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: Dependency Inversion Principle

The following files are used in this section. Download the java source and examine the code.

```
.
├── Button.java
└── Lamp.java
```

1. Which class depends on the other?

2. Suppose we wanted to add a `Fan` class that can be turned on and off to this design and a `Fan` also needed a button.
   (a) Describe the problems that may occur.

   (b) Suppose we handle this by adding 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`)? Draw the diagram.

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

4. Suppose we define an `Equipment` interface for the common behaviour 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:

6. Refactor the code to reflect your final design in an individual project repository.

## Section B: Multiple Principles

1. Explain how the Open/Closed and the Liskov Substitution principles are automatically satisfied when the Dependency Inversion principle is adhered to.

*This is achieved by introducing interfaces for which you can provide different implementations. The interface itself is closed for modification, and you can easily extend it by providing a new interface implementation (satisfies the open closed principle). As long as you don't change any interface abstractions, you can also change higher-level and lower-level components without affecting any other classes, (satisfies the Liskov substitution principle).*

### Part A - SOLID, UML

1 You have been asked to redesign the software architecture for a device emulator that models:

- inkjet printers
- laser printers
- scanners
- multifunction printers

All printers can print documents. An inkjet printer allows fast prints. A laser printer is used for quality prints. Scanners are used to scan documents. Inkjet and Laser printers can produce colour prints. Multifunction printers can print, scan and copy documents. Multifunction printers typically use the same technology and offer identical functionality as laser printers. The Canon MX350 is a concrete kind of multifunction printer.

Suppose the approach shown in Figure 1 was used by a former developer.

```
public interface Device{
    boolean print();
    boolean scan();
    boolean copy();
}
public abstract class InkjetPrinter implements Device{
    boolean fastPrint(){ //code for 1 high speed print }
    boolean colourPrint(){ //code for 1 colour print }
    boolean print(){ // code for 1 regular inkjet print }
    boolean scan() {return false;}
    boolean copy() {return false;}

}
public abstract class LaserPrinter implements Device{
    boolean qualityPrint(){ //code for 1 high quality print }
    boolean colourPrint(){ //code for 1 colour print }
    boolean print(){ // code for 1 regular laser print }
    boolean scan() {return false;}
    boolean copy() {return false;}
}
public abstract class MultiFunctionPrinter implements Device{
    boolean qualityPrint(){//code for 1 high quality print }
    boolean print(){ // code for 1 regular laser print }
    boolean scan() { // code for 1 scan }
    boolean copy() { // code for copying and printing}
}
public abstract class Scanner implements Device{
    boolean print(){return false;}
    boolean scan() { // code for 1 scan }
    boolean copy() {return false;}
}
```

**FIGURE 1**

(a) Identify **four** specific problem(s) with the approach shown in Figure 1, for specifying the objects' behaviours in the scenario. 4 marks

- Classes forced to implement methods that don't apply to them - a lot of useless code (IS)
    - InkjetPrinter, LaserPrinter -> scan( ).
    - Scanner -> print( ), copy( )
- Repeated code across classes . - difficult to maintain consistency (SR) (LS)
    - MultiFunctionPrinter & Scanner -> scan( )
    - MultiFunctionPrinter & LaserPrinter -> qualityPrint( ), print( )
- Can't instantiate the classes (abstract) - there is a concrete MX350 class not implemented
- All devices do not have all of the functionality warranted by the Device interface (LS); (DI)
- Copy method is potentially doing too much - copying and printing (SR)
- MultiFunction printer misses the colourPrint( ) method (LS)

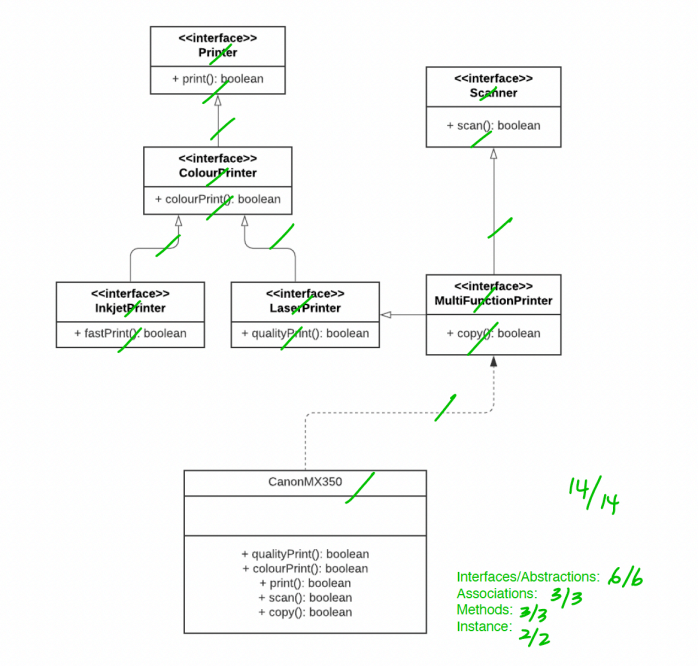
(b) Which **two** SOLID object-oriented design principles are violated in Figure 1? 2 marks

- Single Responsibility
- Liskov Substitution
- Dependency Inversion
- Not open-closed (no discussion about extending the classes)
- Interface segregation

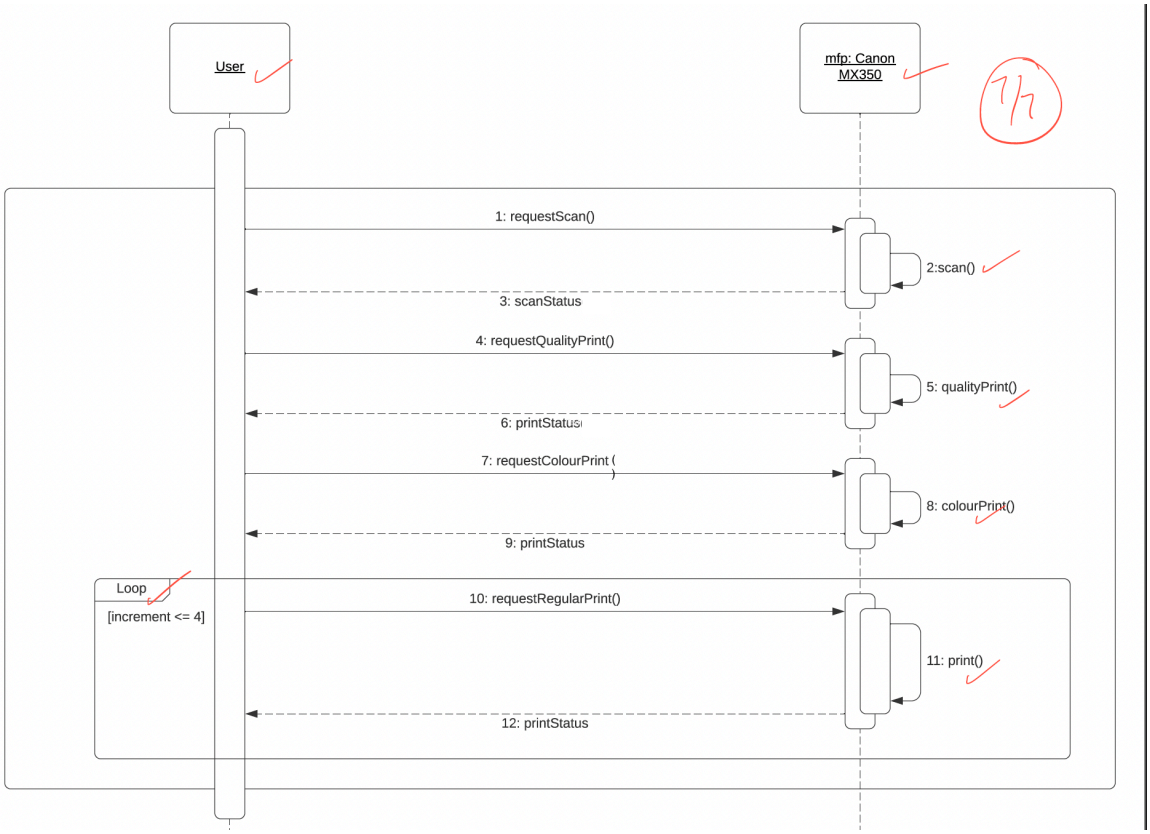
(c) Suggest clearly how the implementation in Figure 1 can be refactored to adhere to the principles identified in part (b) and repair all of the problems listed in part (a). 8 marks

- Split the Device interface into three types:
  - Scanner: scan( )
  - Printer: print( )
  - Copier: copy( )
- Edit abstract classes to only implement the appropriate interface with subclasses, and let concrete classes extend and fill out the code OR introduce additional interfaces and move the code into concrete classes.
- InkjetPrinter, LaserPrinter should implement Printer
- MultifunctionPrinter should implement Scanner, Copier and LaserPrinter
- Introduce a concrete MX350 subclass of MultiFunctionPrinter

(d) Draw a UML class diagram to model a refactored design for the domain in the scenario based on your answers in part (c). You may use the method signatures in Figure 1 and rename classes/ interfaces as necessary. 14 marks




(e) Draw a sequence diagram, using your refactored design, to illustrate the interactions when a user scans a document and prints 1 high quality, 1 colour and 4 regular laser prints with an MX350 device. 7 marks




**TOTAL: 35 MARKS**