National University of San Agustin
System Engineer School

TO - Lab 06 - Solid Principles: Single Responsibility and Open-Closed

Christian E. Portugal-Zambrano

October 30, 2018

1 Solid Principles

S.O.L.I.D is an acronym for the first five object-oriented design(OOD) **principles** by Robert C. Martin, popularly known as Uncle Bob.
These principles, when combined together, make it easy for a programmer to develop software that are easy to maintain and extend. They also make it easy for developers to avoid code smells, easily refactor code, and are also a part of the agile or adaptive software development. This principles are:

- Single Responsibility.
- Open-Closed.
- Liskov Sustitution.
- Interface Segregation.
- Dependency Inversion.

2 Single Responsibility

One of the main concepts stats that:

A CLASS SHOULD HAVE ONE AND ONLY ONE REASON TO CHANGE, MEANING THAT A CLASS SHOULD HAVE ONLY ONE JOB.
So, what it means on code?

```
public class List{
    public List(){
        //building the list
    }

    public void insert(){
        //inserting into the list
    }

    public String print(){
        // printing the list
        return new String("printing");
    }
}
```

In this code the class List have two responsabilities, one is for manage the list and other is for printing the values into a console o file, is this good? We could separate the responsabilities through interfaces or classes, in the next code we define a Printer designed only for Lists.

```
public interface ListPrinter{
    void print();
}
```

The we present the class List but only for one responsability

```
public class ListSR{
    public ListSR(){
        //building the list
    }

    public void insert(){
        //inserting into the list
    }
}
```

So in this way if we want just a GraphViz printer we just have to extend it from the interface

```
public class ListGraphVizPrinter implements ListPrinter{
    ListGraphVizPrinter(ListSR _list){

    }

    public void print(){
        System.out.println("Printing␣the␣list␣using␣GraphViz");
    }
}
```

Then, our demo is:

```java
public class SingleResponsibility {
   public static void main(String [] args){
      ListSR myList = new ListSR();
      ListPrinter printer = new ListGraphVizPrinter(myList);
      printer.print();

   }
}
```

consequently we can extend our code to implement as much as we need printer classes.

3 Open-Closed

It says that:

A class should be closed for modifications and open for extensions. In other words, we should extend class functionality without modifying its core behavior. We can achieve this by:

- using inheritance, adding structure to the new classes
- using composition, the black box keeps encapsulation protected.
- applying Dependency Injection, promote polymorphism to achieve these
- applying Decorator pattern, we will see this later.
- applying Strategy pattern, we will see this later.

This principle is frequently attached to all our code, here at List code presented above we can see that the class List is a class to instances List objects but as we don't use interfaces, to achieve more functionality we need to change the internal structure of list, this List class has high cohesion and high coupling and violates the open-close principle, but in ListSR and List Printer interface we promote the reutilization through inheritance and polymorphism. The only idea behind this principle is reutilization and extension.

4 Warm up

Base on the principle showed above, you have to modify your code of structures to accomplish these principles, the main requirement is add a printer functionality to save the structures on different formats, you can use the knowledge acquired from practice 05 where VTK shown the use of different formats.

5 To do

We need to keep programming, a framework that implement the single responsibility principle is the java persistence API (JPA), as the documentation of it says:

THE JAVA PERSISTENCE API PROVIDES A POJO PERSISTENCE MODEL FOR OBJECT-RELATIONAL MAPPING.
Then only one responsibility is given to these model. We need to consider another approach which extends JPA, Hibernate is an object-relational mapping tool for the Java programming language. It provides a framework for mapping an object-oriented domain model to a relational database. We can say that hibernate use and extend the JPA functionality, the focus of this work is to implement a JPA application that use the data structures programmed until now, in other words you have to add a persistence functionality to the structures using JPA or Hibernate. Consider mainly the two approach related in this practice to make the implementation.

6 Deep inside

This section is just for anyone who wants to make a deep inside into the theory and like challenges[1], you will have to prepare a presentation of just 5 minutes to explain and run the code, then you will have to defend yourself another five minutes of questions. I want that all students benefit from your presentation, remember that we are here to learn. If you want to do this please email to cportugalz@unsa.edu.pe

7 Deadline

This report will be qualified on November, Tuesday 13 or Wednesday 14. Remember that plagiarism will be punished. All question and doubts must be done to the email.

References

[1] R. C. Martin, *Clean architecture: a craftsman's guide to software structure and design.* Prentice Hall Press, 2017.

[1]This must not be included into the report