course: COMP6215 week: 04 lesson: 02 topic: solid-5-and-assignment-prep

DOWNLOAD PDF CLICK HERE

SOLID Principles

Each week in this section of the course we will look at 1 or 2 of the SOLID principles.

The principles stand for:

- Single Repsonsibility Principle
- Open Closed Principle
- Liskov Substitution Principle
- Interface Segregation Principle
- **Dependency inversion principle**

The idea behind using this principles is that you look at how you strucutre and organise your code so that it is easier to maintain. Although it will look like you need to do more work (and it is) for things that are relatively simple, it will help you to read and update your code better once it is in production.

The other takeaway from these principles is that you are going to go through a process of decoupling your code from the other code in your application.

PART 2: - Solid Summary

Over the last few weeks we have looked at the SOLID principles, let's have a look at a brief overview of what they mean:

1. SRP - Single Responsibility Principle

> A class should have one, and only one, reason to change.

2. OCP - Open Closed Principle

> “Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification.”

3. LSP - Liskov Substitution Principle

> When you define the base class as a type, then you should be able to instantiate it with a sub class and not break your code You should also stick to the original definition of your methods and not change the input and outputs of them

4. ISP - Interface Segregation Principle

> “Clients should not be forced to depend upon interfaces that they do not use.”

5. DIP - Dependency Inversion Principle

> High-level modules should not depend on low-level modules. Both should depend on abstractions. Abstractions should not depend on details. Details should depend on abstractions.

Now we have seen that these principles carry different levels of complexities, but now look at each one of them and apply an interface to them.

Interfaces do not solve all of the problems, but because they enforce a contract which removes any dependencies and allow for classes to be clearly defined so they can have a cleaner implementation.