course: COMP6210 week: 01 lesson: 02 topic: solid-principle-1-single-responsibility-principle

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:

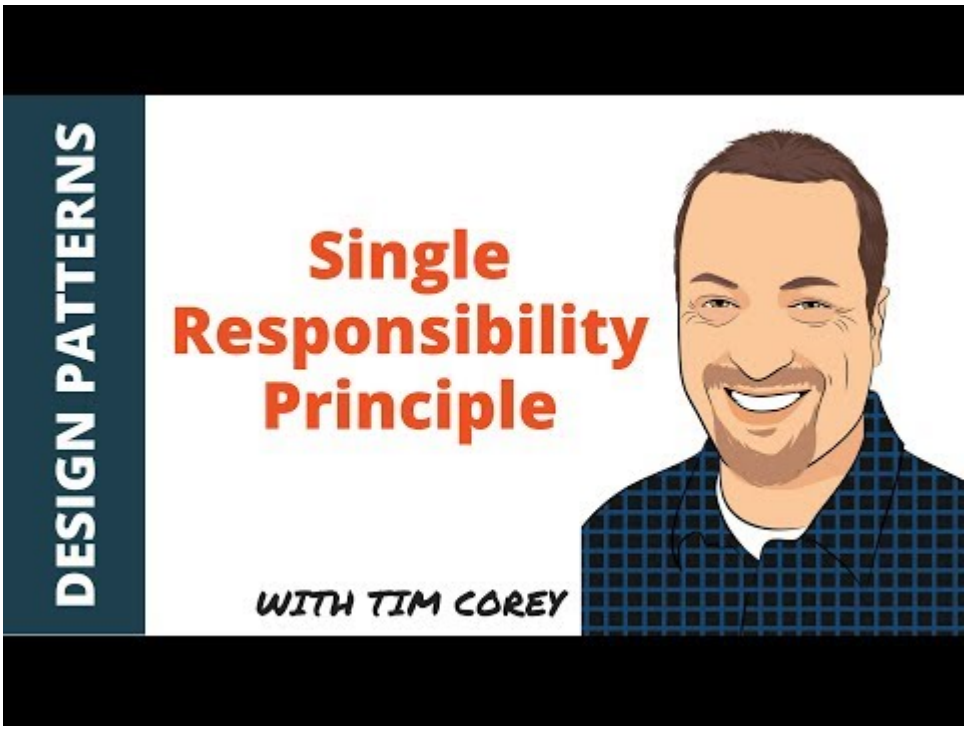
- Single Repsonsibility Principle
- Open Closed Principle
- Liskov Substitution Principle
- Interface Segregation Principle
- Dependency inversion principle

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 Principle 1 - Single Responsibility Principle

Watch a video about it here




The Single responsibility principle is the simplest one of all of them and can be summed like this:

Each class and method should only be responsible for a single thing. As soon as it does 2 or more things, it should be split up into multiple classes and methods respectively.

A couple of "guidelines" you can quickly see if you are breaking the rule

- Is your method longer than 10 lines?
- Has your class got more than 10 things in it?
- Do you have more than a single code indentation in your method?

Let's have a look at the following code:

```
class Program
{
  private void Main(string[] args)
  {
    Console.WriteLine("What is your name?");
    string name = Console.ReadLine();
    Console.WriteLine($"Your name is {name}.");
  }
}
```

Is it very simple and not a lot happens, but let's look at a number of details:

- Everything happens in a single method
- The method actually does 3 things
- You cannot change the output of the program without changing the main program

The `Program` class is responsible for delivering the program to the user, but it shouldn't hold any data - data is only contained in this class to pass it back to a more permanent storage.

The data displayed in the `Console.WriteLine` method should come from a static variable. The reason for this is so that we can reuse it later in another program.

So we could do something like this:

```
class Program
{
  private static string question = "What is your name?";

  private void Main(string[] args)
  {
    Console.WriteLine(question);
    string name = Console.ReadLine();
    Console.WriteLine(Greeting(name));
  }

  private static string Greeting(string name)
  {
    return $"Your name is {name}";
  }
}
```

This is much better, since we have identified that the `Main` method does only one thing - display and get information from - or interact with - the user.

However the Program class still does more than a single thing and the static variable and the method are not strictly related to the `Program` class. We could extract those into a class called `Messages`. So let's do that:

```
//Program.cs File
class Program
{
  private void Main(string[] args)
  {
    Console.WriteLine(Messages.question);
    string name = Console.ReadLine();
    Console.WriteLine(Messages.Greeting(name));
  }
}

//Message.cs File
class Messages
{
  public static string question = "What is your name?";

  public static string Greeting(string name)
  {
    return $"Your name is {name}";
  }
}
```

The code above conforms to the Single Responsibility Principle.

The idea is that each class has a single purpose and that each method as a single purpose. If you look at your code that you are working on and match it to the guidelines above then you can quickly see if you can break your code down into smaller pieces.

Smaller pieces might mean more code to write, but it is easier to maintain and easier to find any bugs.