**SOLID Principles:**

These Principles are used in OOD(Object Oriented Design) which helps us to keep the code

* modular
* maintainable
* less complex
* easy to understand
* avoid duplication

**Acronym of SOLID**

S - Single Responsibility Principle

O - Open/Closed Principle

L - Liskov Substitution Principle

I - Interface Segregation Principle

D – Dependency Inversion Principle

**Single Responsibility Principle:**

* This principle states that **each class should have** **one responsibility, one single purpose**. This means that a class will do only one job, which leads us to conclude it should have **only one reason to change**
* Any Method/Function/Class which we create should have one responsibility and should only have one reason to change

Advantages:

* Adding multiple functionalities to single class can lead to complexity and hard to maintain
* Changing one functionality can effect other functionalities because they exist in a single class

**Examples:**

**Example 1:** We need to develop a Feature where we can manipulate a given text in different ways and Print the text in different ways

**Solutions:**  This feature can be developed in many ways, mentioning some below:

Approach 1: Developing both the features in a single place

Approach 2: Break Down the Feature into multiple parts where each part does one job

* One Class does the job of Text Manipulating
* One Cass does the job of Print the text

***Approach 2 will follow Single Responsibility principle***

**Example 2:** We need to develop a Feature where we need to add some Invoice and delete the invoices, generate reports and email reports

**Solutions:** This feature can be developed in many ways, mentioning some below:

Approach 1: Developing both the features in a single place

Approach 2: Break Down the Feature into multiple parts where each part does one job

* One Class for add and remove invoice
* One Class for generating Report
* Once Class for Sending email of report

***Approach 2 will follow Single Responsibility principle***

**Example 3: Feature to develop to compile a text and print the text**

**Solution:**

consider a module that compiles and prints a report. Imagine such a module can be changed for two reasons. First, the content of the report could change. Second, the format of the report could change. These two things change for different causes. The single responsibility principle says that these two aspects of the problem are really two separate [responsibilities](https://en.wikipedia.org/wiki/Interface_(computing)), and should, therefore, be in separate classes or modules. It would be a bad design to [couple](https://en.wikipedia.org/wiki/Coupling_(computer_programming)) two things that change for different reasons at different times.

The reason it is important to keep a class focused on a single concern is that it makes the class more robust. Continuing with the foregoing example, if there is a change to the report compilation process, there is a greater danger that the printing code will break if it is part of the same class.