**Solid Principle**

They allow us to create software that is understandable, can be understood and flexible, leaving it as concrete and simple as possible.

The following five concepts make up our SOLID principles:

1. **S**ingle Responsibility

This principle states that **a class should only have one responsibility. Furthermore, it should only have one reason to change.**

**Benefits:**

1. **Testing** – A class with one responsibility will have far fewer test cases.
2. **Lower coupling** – Less functionality in a single class will have fewer dependencies.
3. **Organization** – Smaller, well-organized classes are easier to search than monolithic ones.
4. **O**pen/Closed

Known as the**open-closed principle.**Simply put,**classes should be open for extension but closed for modification. In doing so, we stop ourselves from modifying existing code and causing potential new bugs**in an otherwise happy application.

Of course, the**one exception to the rule is when fixing bugs in existing code.**

1. **L**iskov Substitution

Simply put, if class A is a subtype of class B, we should be able to replace B with A without disrupting the behavior of our program.

1. **I**nterface Segregation

Stands for interface segregation, and it simply means that **larger interfaces should be split into smaller ones. By doing so, we can ensure that implementing classes only need to be concerned about the methods that are of interest to them.**

1. **D**ependency Inversion

**The principle of dependency inversion refers to the decoupling of software modules. This way, instead of high-level modules depending on low-level modules, both will depend on abstractions.**

**BEST PRACTICES, PATTERNS, AND IDIOMS**

* Best practices are procedures or techniques that help developers adhere to principles, without having to consider the details of a situation at a theoretical level.
* Patterns exemplify principles, by providing proven solutions to reoccurring problems in specific contexts.
* Idioms are techniques or solution for expressing a certain algorithm or data structure in a specific programming language, in a way that is consistency with certain principles.

**Abstraction:** An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer

**Encapsulation**: We encapsulate the details of a class inside its private part so that it is impossible (and not just suggested) for any of the class's clients to know about or depend upon these details.

The ability to change the representation of an abstraction (data structures, algorithms) without disturbing any of its clients is the essential benefit of encapsulation.

**Modularity**: Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules

Basic idea: Partition a system such that parts can be designed and revised independently (“divide and conquer”) System is partitioned into modules, with each one fulfilling a specific task Modules should be changeable and reusable independent of other modules