* There will be 5 solid principles
* If we follow those principles, we can avoid the most of the software design problems
* This will make applications more understandable, easy maintainable
* To avoid tightly coupled and little encapsulation and all
* **S: Single Responsibility Principle (SRP)**
* **O: Open closed Principle (OSP)**
* **L: Liskov substitution Principle (LSP)**
* **I: Interface Segregation Principle (ISP)**
* **D: Dependency Inversion Principle (DIP)**

**Single Responsibility Principle (SRP)**

* SRP says, "Every software module should have only one reason to change".
* This means that every class, or similar structure, in your code should have only one job to do.
* Everything in that class should be related to a single purpose

**Open closed Principle (OSP)**

* A software module/class is open for extension and closed for modification
* Here "Open for extension" means, we need to design our module/class in such a way that the new functionality can be added only when new requirements are generated.
* "Closed for Modification" means we have already developed a class and it has gone through unit testing.
* We should then not alter it until we find bugs. As it says, a class should be open for extensions; we can use inheritance to do this

**Liskov substitution Principle (LSP)**

* You should be able to use any derived class instead of a parent class and have it behave in the same manner without modification
* It ensures that a derived class does not affect the behavior of the parent class, in other words, that a derived class must be substitutable for its base class
* We must ensure that new derived classes extend the base classes without changing their behavior

**Interface Segregation Principle (ISP)**

* Instead of one fat interface, many small interfaces are preferred based on groups of methods based on requirements
* An interface should be more closely related to the code that uses it than code that implements it
* Each interface should have a specific purpose/responsibility (refer to SRP).
* You should not be forced to implement an interface when your object doesn't share that purpose

**Dependency Inversion Principle**

The Dependency Inversion Principle (DIP) states that high-level modules/classes should not depend on low-level modules/classes. Both should depend upon abstractions





