So, **SOLID principle** will help us to write loosely coupled code which is highly maintainable and less error prone.

**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)**

S: 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.

O: Open for extension means, you should be able to extend a class’s behaviour, without modifying it

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.

Solution: can be done using overriding and inheritance.

L: The Liskov Substitution Principle (LSP) states that "you should be able to use any derived class instead of a parent class and have it behave in the same manner without modification".

A derived class must be substitutable for its base class.

Solution: we can fix this problem using the Interface segregation principle by (ISP) by identifying the abstraction and the responsibility separation method using proper interfaces.

I: Interface Segregation Principle (ISP) An interface should be more closely related to the code that uses it than code that implements it. So the methods on the interface are defined by which methods the client code needs than which methods the class implements.

So clients should not be forced to depend upon interfaces that they don't use.

Solution: Separate responsibilities/purposes and distribute them on multiple interfaces which provides a good level of abstraction too.

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

Solution: To do that we need to introduce an abstraction between them, so that high-level class can contact the abstraction to log the exception instead of depending on the low-level classes directly.