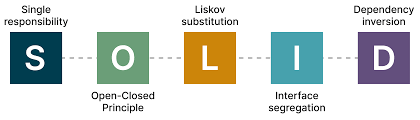
1. **What is SOLID?**

***SOLID*** is an acronym for the first five object-oriented design (OOD) principles by Robert C. Martin, known as "Uncle Bob."



Those design principles can be seen as guidelines for any object-oriented design that when followed, help produce maintainable and extensible code.

The following is a list of those design principles that make the acronym "SOLID":

1. **S** - **S**ingle **R**esponsibility **P**rinciple (**SRP**) = A class should have one, and only one, reason to change.
2. **O** - **O**pen/**C**losed **P**rinciple (**OCP**) = Entities should be open for extension, but closed for modification.
3. **L** - **L**iskov **S**ubstitution **P**rinciple (**LSP**) = Derived classes must be substitutable for their bas e classes.
4. **I** - **I**nterface **S**egregation **P**rinciple (**ISP**) = A client should not be forced to implement and interface that it doesn't use.
5. **D** - **D**ependency **I**nversion **P**rinciple (**DIP**) = = High-level modules shouldn’t depend on low-level modules instead on abstractions, not on concretions.

## What happens if we don't use SOLID?

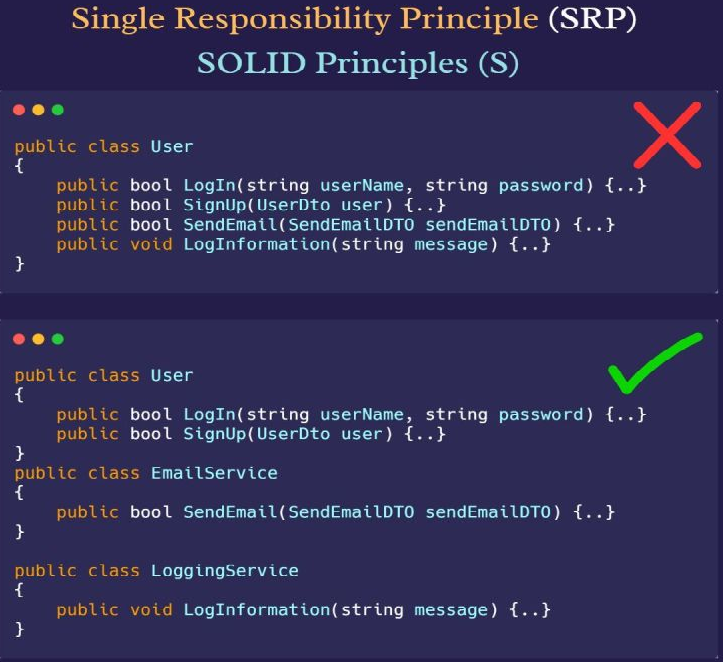
* Tight coupling occurs
* Duplication of code
* Unknown issues in application development lifecycle
* requires time to do modification in code
* code not testable

## Why use SOLID?

* Reduces complexity of code.
* Reusability
* Reduces Errors
* Removes Duplication
* Increase Readability, Extensibility
* Better Testability
* Reduces Tight Coupling

**1.1 What is SRP Principle of SOLID, explained with an example?**

SRP says:-  
- A class/method should have one reason to change  
- A class/method should have only one responsibility  
- A class/method should be related to a single purpose  
  
With SRP what we achieve:-  
- Class/method becomes smaller and cleaner  
- Now in future updating the code becomes easier  
  
Example  
  
Bad: - Single class responsible for Signup, Login , Logging Error and Sending the Email.  
  
Good :- Different classes for different purpose  
User Account related (Signup and Login)  
Sending Email related (Send Email)  
Logging related (Log Information)  
  
Now in future we can add Log Error, Log Warning in our logging class and ForgotPasword in User Account.  
  
Suppose if we had kept a single class and we add new methods in that class then its impact would have been everywhere  
where class has dependencies.  
  
In current example I have used just class for demo one can use interface example as well.

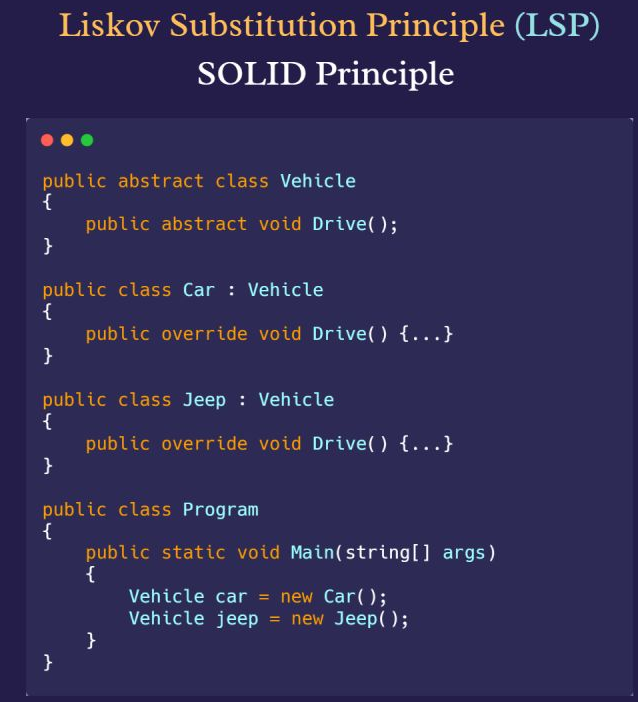


**1.2 What is the Open/Closed Principle of SOLID, explained with an example?**

OCP is the second principle of SOLID, the most important one and it is linked with SRP.  
  
The Open/close Principle says:-   
1) Classes/methods/modules should open for extension but closed for modification  
  
2) New functionality should be implemented by adding new classes and methods instead of changing the existing class and their methods  
  
What if I don't follow OCP :-   
- We break SRP  
- Effects on another part of the code  
- Difficult to maintain code  
- More testing, more headache  
- Costly in time/cost and resources.  
  
Multiple ways to implement OCP:-   
- Using an abstract class  
- Using interfaces  
  
Example   
Bad:- A single method of class responsible for calculating basic salary, when a new requirement comes we add a new if condition and solve it  
  
Good:- A base abstract class with the abstract method and children class override it according to their needs

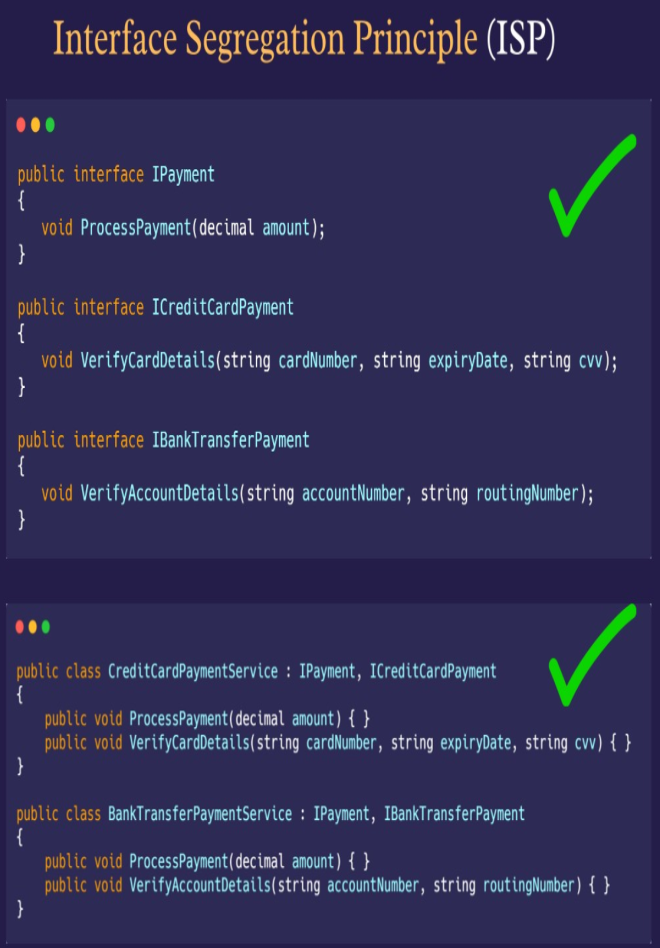
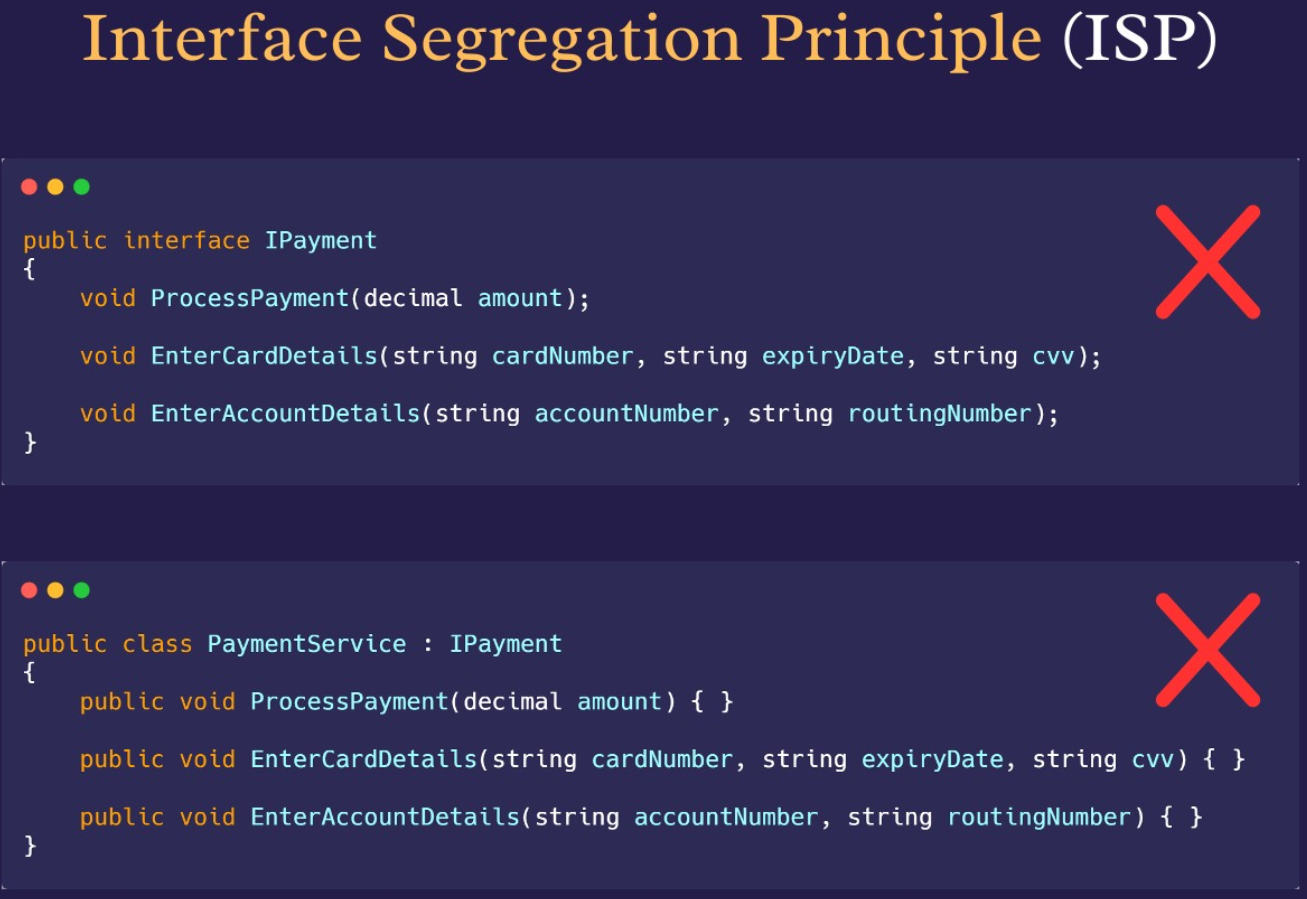
** **

**1.3 What is Liskov Substitution Principle, how to implement it?**  
**LSP** is third principle of SOLID; it is extension of open/close principle.  
  
**LSP** says :-   
- Derived class must be correctly substitutable for base class  
- Subtypes must be substitutable for their base type (interface, class)  
- Sub class should be able to replace objects of base class without affecting  
  
**LSP** tell us how to use inheritance properly , if you see somewhere in your code NotImplementedException then it is most probably violation of LSP.  
  
**Benefits of LSP:-**   
- Code reusability  
- More maintainable code  
- Better use of inheritance  
  
**Example**  
**Bad**:- A base class with some methods and sub classes throwing NotImplementedException for those methods it don't need  
  
**Good**:- A base class with some methods and sub classes implement those methods and those are needed as well.

****

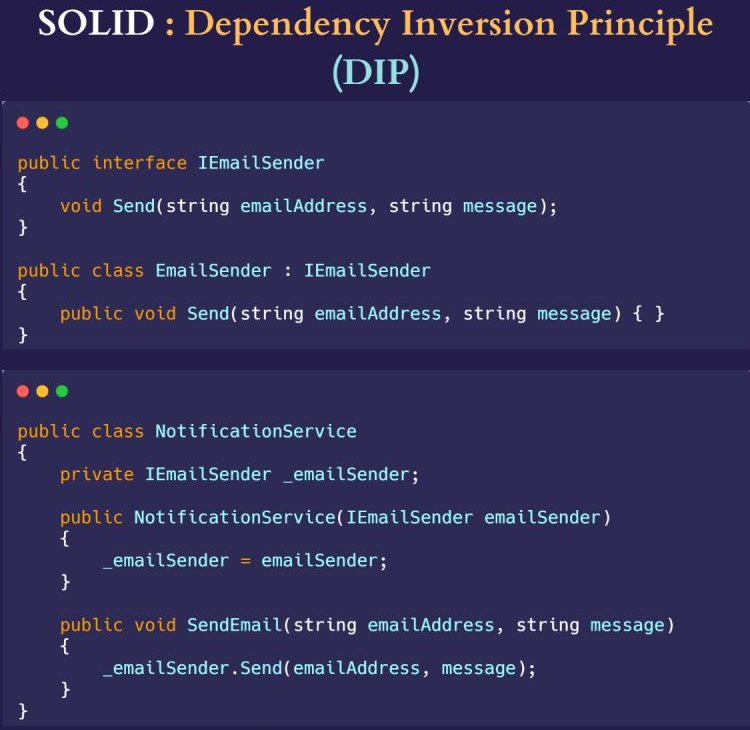
**1.4 What is Interface Segregation Principle and how to implement it?**

ISP is fourth and second last principle from SOLID principles. At some point ISP and SRP resemble.  
  
ISP says :-   
- Class should not be forced to depend on methods it doesn't use  
- Split a lengthy interface to small relevant interfaces  
  
Benefits of ISP:-  
- Modularity  
- Reusability  
- Maintainability  
- Low coupled code  
- Reduces the chances of defects   
- Makes code easy to test and deploy independently  
  
Example   
Bad :- Single interface that is responsible card and bank payment processing even if someone would be interested in only one  
  
Good :- Divide single interface to three different interfaces one to process payment and other two for bank and card.

****

**1.5 What is dependency inversion principle** **and how to implement it?**

Dependency Inversion Principles is last one in SOLID principles and most important one it says, "High level module should not depend on low level module, instead both should depend on abstractions"  
  
Benefits   
- Decoupling  
- Testability  
- Reusability  
  
Multiple ways to implement DIP  
- Constructor injection  
- Property injection  
- Method injection  
  
Example   
Bad: Instead of using abstraction one using direct instantiation using new keyword everywhere.  
  
Good : Relying on abstraction , registering dependency and using it via constructor injection (Method and Property injection as well)



**Examples**

**SRP**: [**https://github.com/r-tarnowski/SingleResponsibilityPrinciple/blob/master/src/SingleResponsibilityPrinciple.cpp**](https://github.com/r-tarnowski/SingleResponsibilityPrinciple/blob/master/src/SingleResponsibilityPrinciple.cpp)

**ORP:** [**https://github.com/r-tarnowski/OpenClosedPrinciple/blob/master/src/OpenClosedPrinciple.cpp**](https://github.com/r-tarnowski/OpenClosedPrinciple/blob/master/src/OpenClosedPrinciple.cpp)

**LSP:** [**https://github.com/r-tarnowski/LiskovSubstitutionPrinciple/blob/master/src/LiskovSubstitutionPrinciple.cpp**](https://github.com/r-tarnowski/LiskovSubstitutionPrinciple/blob/master/src/LiskovSubstitutionPrinciple.cpp)

**ISP:** [**https://github.com/r-tarnowski/InterfaceSegregationPrinciple/blob/master/src/InterfaceSegregationPrinciple.cpp**](https://github.com/r-tarnowski/InterfaceSegregationPrinciple/blob/master/src/InterfaceSegregationPrinciple.cpp)

**DIP:** [**https://github.com/plusangel/DependencyInversion\_pattern\_cpp/blob/master/di\_approach.cpp**](https://github.com/plusangel/DependencyInversion_pattern_cpp/blob/master/di_approach.cpp)

**References**

<https://platis.solutions/blog/2020/06/22/how-to-write-solid-cpp/>

<https://www.freecodecamp.org/news/solid-principles-explained-in-plain-english/>

<http://codingadventures.org/2021/11/27/solid-principles-using-modern-c/>

<https://cpp-design-patterns.readthedocs.io/en/latest/principles.html>

<https://github.com/plusangel/solid_principles_cpp>

<https://github.com/tal95shah/SOLID_Principles>

<https://github.com/rhazari/SOLID-Principles>

<https://github.com/ArjunSingh13/SOLID-Principes-CPP>

<https://github.com/Sudhanshu-Gupta05/StockTrading>

6 essential clean code tips to become better Software Engineer:  
- Indentation.   
- Meaningful names.  
- Avoid passing NULLs.  
- **KISS (**keep it simple stupid).  
- Command query separation.  
- Keep methods and classes small.