Software Design Principles (S.O.L.I.D)

- ✓ Are you a software developer/researcher/teacher?
- ✓ Do you write poor-quality software, and you don't know what are your deficiencies exactly?
- ✓ Have you read the SOLID principles, but you forget them or cannot explain them clearly?
- ✓ Have you wondered about the direct relation between SOLID principles and OOP concepts or design patterns?

Then, this **capsule** is for you! You can now find all the answers in **one table**.

Caveat:

A reader should be familiar with SOLID and OOP principles before reading this capsule summarization. It is not meant to explain these principles from scratch. This is free to print or distribute as is.

Let me know in the comments below if there is a difficult/challenging concept you would want to see a summarization table for.

Principle	**S**ingle Responsibility (SRP)	**O**pen/Closed (OCP)	**L**iskov Substitution (LSP)	**I**nterface Segregation (ISP)	**D**ependency Inversion (DIP)
What is the meaning?	- A class or method should have one purpose only. Example	A class should be open for extension but closed for modification. Example	Example	Example	Example
How to achieve it?	- By making one class to do one thing and do it well. - By making one method to do one thing and do it well.	-Split a responsibility of class/method in a way that the class/method can be extended or replaced but not modified. -Use Inheritance concept to make use of a class.			
When is it violated?	- When a class is very long. - When a class does 2 or more different responsibilities. - When a method does 2 or more different tasks.	- When a class/method must be modified to add a new feature. (Class is open for modification not extension) -			
How to solve the violation?	-Split one class/method into 2 or more logical classes/methods.				
-Why do we need it? -What is the benefit? -How to achieve such benefit?	1. Promotes **Robustness,** by not breaking the code once a class/method is changed. 2. Promotes **Maintainability**, by having only one reason to change a class/method. 3. Promotes **Usability**, by having a specific need to use a class/method.	1. Promotes **Extensibility**, by inheriting from a class and add a new feature/use case to it without modifying existing code. 2. Promotes **Maintainability**, by replacing a single class/method and not modify it, because sometimes it is compiled/distributed and hence unmodifiable. 3. Promotes **re-utilization**, by using existing functionalities in a class and not re-implement them again.	- model good inheritance hierarchies but why?		- Reduces dependencies
What OOP concept is leveraged?	- **Encapsulation** How? - **Abstraction** How?	**- Encapsulation** How? **- Abstraction** How? **- Inheritance** How? **- Polymorphism** How?			
What Design patterns is leveraged?	- KISS – Keep It Simple Stupid - DRY – Don't Repeat Yourself	- DRY – Don't Repeat Yourself			DRY – Don't Repeat Yourself

Table 1 – SOLID Principles and its relation to OOP and design patterns.

Author: **Mohamed H. Oreaba** | Head of Software Engineering | M.Sc. in Computer Science | The American University in Cairo | **mhamdy@aucegypt.edu**

Object Oriented Programming Concepts (O.O.P)

OOP Concept	Encapsulation	Abstraction	Inheritance	Polymorphism
Meaning				
How to achieve it?				
How to violate it?				
How to solve the violation?				
What is the benefit? And How?				
What Design patterns is leveraged?				DRY – Don't Repeat Yourself