SOLID Principles

The SOLID Principles are five principles of Object-Oriented class design. They are a set of rules and best practices to follow while designing a class structure.

The SOLID principles were first introduced by the famous Computer Scientist Robert J. Martin (a.k.a Uncle Bob) in his paper in 2000. But the SOLID acronym was introduced later by Michael Feathers.
Uncle Bob is also the author of bestselling books *Clean Code* and *Clean Architecture*, and is one of the participants of the "Agile Alliance".
Therefore, it is not a surprise that all these concepts of clean coding, object-oriented architecture, and design patterns are somehow connected and complementary to each other.

They all serve the same purpose:

"To create understandable, readable, and testable code that many developers can collaboratively work on."

Let's look at each principle one by one. Following the SOLID acronym, they are:

- The **S**ingle Responsibility Principle(SRP).
- The **O**pen-Closed Principle(OCP).
- The **L**iskov Substitution Principle(LSP).
- The **I**nterface Segregation Principle(ISP).
- The **D**ependency Inversion Principle(DIP).

The Single Responsibility Principle

The Single Responsibility Principle states that **a class should do one thing and therefore it should have only a single reason to change**.
To state this principle more technically: Only one potential change (database logic, logging logic, and so on.) in the software's specification should be able to affect the specification of the class.

Open-Closed Principle

The Open-Closed Principle requires that **classes should be open for extension and closed to modification.**
Modification means changing the code of an existing class, and extension means adding new functionality

Liskov Substitution Principle

The Liskov Substitution Principle states that subclasses should be substitutable for their base classes.

Interface Segregation Principle

Segregation means keeping things separated, and the Interface Segregation Principle is about separating the interfaces.

The principle states that many client-specific interfaces are better than one general-purpose interface. Clients should not be forced to implement a function they do no need.

Dependency Inversion Principle

The Dependency Inversion principle states that our classes should depend upon interfaces or abstract classes instead of concrete classes and functions.

In his article (2000), Uncle Bob summarizes this principle as follows:
"If the OCP states the goal of OO architecture, the DIP states the primary mechanism".

These two principles are indeed related and we have applied this pattern before while we were discussing the Open-Closed Principle.