SOLID Software Design Principles

|  |  |  |
| --- | --- | --- |
|  | PRINCIPLE | DESCRIPTION |
| S | Single Responsibility | A class should have one, and only one, reason to change, which means it should have only one function. |
| O | Open/Closed | Software objects should be open to extension, but closed for modification. |
| L | Liskov Substitution[[1]](#footnote-1) | Objects of the same type should be replaceable with others from the same category without altering the function of the program.[[2]](#footnote-2) |
| I | Interface Segregation | No client should be forced to depend on methods it does not use. The program’s interfaces should always be kept smaller and separate from one another. |
| D | Dependency Inversion | High-level modules[[3]](#footnote-3) should not depend on low-level modules, but both should depend on abstractions. While abstractions should not depend on details[[4]](#footnote-4), details should depend on abstractions. |

1. **Covarianza:** El tipo de retorno de un método: si tienes un tipo de retorno, ese tipo no puede cambiar.

   **Contravarianza:** El tipo de entrada de un método: si tienes un tipo de entrada, ese tipo no puede cambiar. [↑](#footnote-ref-1)
2. If **S** is a subtype of **T**, then objects of type **T** may be replaced with objects of type **S** without breaking the program. [↑](#footnote-ref-2)
3. In this instance, a high-level module means anything that’s calling something else. [↑](#footnote-ref-3)
4. “Should not depend on details” means abstraction shouldn’t have to know about how things get done, i.e., a really clear interface, so interfaces are a clear way of doing DI (Dependency Inversion). [↑](#footnote-ref-4)