SOLID Design Principals

Wednesday, November 23, 2022 9:12 AM

The SOLID design principles help us create maintainable, reusable, and flexible software designs. Each letter in the acronym SOLID stands for a specific principle.

Here is what each letter in the acronym stands for:

S: Single responsibility principle.
O: Open–closed principle.
L: Liskov substitution principle.
I: Interface segregation principle.
D: Dependency inversion principle.

Single responsibility principle (SRP)

Every class, module, or function in a program should have **one responsibility/ purpose** in a program.
Every class should have **only one reason to change**.

Open-closed principle (OCP)

Software entities should be **open for extension**, but **closed for modification**. Such entities - classes, functions, and so on- should be created in a way that their core **functionalities** can be **extended** to other entities **without altering the initial entity's** source code.

Liskov substitution principle (LSP)

The Liskov substitution principle states that:
Let Φ(x) be a property provable about objects x of type T. Then Φ(y) should be true for objects y of type S where S is a subtype of T.

This principle implies that when an **instance of a class is passed**/ extended to **another class**, the **inheriting class** should have a **use case** for **all the properties** and **behavior** of the **inherited class**.

Interface segregation principal (ISP)

The interface of a program should be split in a way that the **user**/ client would **only have access to the necessary methods** related to their needs.

Liskov substitution principle **X** Interface segregation principle

LSP: when a new class has the need to inherit an existing class, it should do so because this new class has a need for the methods the existing class has.
ISP: it is unnecessary and unreasonable to create an interface with a lot of methods as some of these methods may be irrelevant to the needs of a particular user when extended.

Dependency inversion principle (DIP)

High-level modules should **not import** anything **from low-level modules**. Both should depend on abstractions.
Abstractions should not depend on details. **Details** should **depend** on **abstractions**.