Mastering the SOLID Principles

Writing understandable, maintainable, and flexible code

Overview

Principle	Name	Purpose
S	Single Responsibility Principle (SRP)	A class should have only one reason to change.
O	Open/Closed Principle (OCP)	Entities open for extension, but closed for modification.
L	Liskov Substitution Principle (LSP)	Subtypes must be substitutable for their base types.
I	Interface Segregation Principle (ISP)	Clients should not be forced to depend on methods they do not use.
D	Dependency Inversion Principle (DIP)	High-level modules should not depend on low-level modules. Both should depend on abstractions.

S - Single Responsibility Principle (SRP)

> Every class should focus on doing only one thing.

A practical example is implemented. Interested users can click here to review the material and get hands-on to the principle.

O - Open/Closed Principle (OCP)

> Open for extension but closed for modification.

In other words, able to add new features without changing old code. An example is provided to compare the effect of using `O-principle` and without using it. In addition, highlighted the concept of abstract method.

L - Liskov Substitution Principle (LSP)

> Subclasses must be substitutable without any changes to their base class.

The dataloader example is continued and being applied each principle step-by-step. The `L-principle` is applied to the taken example. A comprison results are presented here. Moreover, a discussion to static method is also provided at the end.

I - Interface Segregation Principle (ISP)

> Class should not depend or contains things they are not required.

Initailly, there is a single file contains CConfigLoader, PathManager, and DotaLoader. While applying different principles step-by-step, the code is becoming lengthly. It is time to split the code based on the performed tasks by the classes. If anything that does not required, remove them. The code is presented in the directory.

D - Dependency Inversion Principle (DIP)

> High-level logic (e.g. `ConfigLoader`) should depend on abstract contracts (`ConfigReader`), not direct classes like `YamlConfigParser`.

With configuration, `PathManager` also supports abstract base class method. A significant changes can be seen in the `DataLoader` as well. The source code can be viewed and analyzed.

Additional information

It is a self-explanatory tutorial on SOLID principle. Any new concept used in the tutoral are discussed in detail.

- Abstract method
- Static method
- Over-segregation

Conclusion

Principle	Summary
SRP	Separated ConfigLoader, PathManager, DataLoader
OCP	Added JSON loader without modifying existing code
LSP	Swap YAML/JSON parsers safely by checking the extension
ISP	Break large interfaces into smaller ones and class segregation
DIP	Inject dependencies via abstraction (not hardcoded)