Liskov Substitution Principle (LSP)

> Subtypes must be substitutable for their base types.

That is, if class B is a substitute of A, it should behave like A when used in place of it.

Already applied in the previous principle. Let us make it automatic.

```
class ConfigParserFactory:
    @staticmethod
    def get_parser(path):
        ext = os.path.splitext(path)[-1].lower()

        if ext in [".yaml", ".yml"]:
            return YamlConfigParser()
        elif ext == ".json":
            return JsonConfigParser()
        else:
            raise ValueError(f"Unsupported extension: {ext}")
```

class B is `YamlConfigParser` and class A is `JsonConfigParser`. Therefore, no matters which parser is going to use, the code behaves the same. The reason behind it is that both class A and B used abstract base class and fulfills the abstract method requirements, i.e., implementation of `load()` function.

Static Method

A `@staticmethod` in Python defines a method that belongs to a class but does not access or modify the class state (cls) or instance state (self). Meaning, it does not take `self` or `cls` as a parameter. Therefore, it cannot access instance or class variable directly. It can be called without creating an instance.

1. Class Context:

   `ConfigParserFactory` is a utility class that acts like a "factory" for creating parser objects based on file extension.

2. Method Definition:

   `@staticmethod` is used to define get_parser(path). It doesn't use `self` or `cls`, because it does not need to.

3. Behavior:

   Based on the file extension, it returns the appropriate config parser object:

   - `.yaml` or `.yml` --> `YamlConfigParser`
   - `.json` --> `JsonConfigParser`

   If unsupported --> raises a ValueError.