Interface Segregation Principle (ISP)

> A client should not be forced to depend on methods it does not use.

In simple words:

- Do not put everything in one big interface (class).
- Instead, break interfaces into smaller, more specific ones that serve a particular purpose.

without ISP_

The below `ConfigParser` interface is minimal and single-purpose. Advantages are:

- Subclasses (`YamlConfigParser` and `JsonConfigParser`) implement only what they need
- Clients (`ConfigLoader`) can rely on `load()` without worrying about how it is implemented.

```
class ConfigParser(ABC):
    @abstractmethod
    def load(self, path):
        pass
```

Considering a modification in interface as follows:

```
class ConfigParser(ABC):
    @abstractmethod
    def load_yaml(self, path): pass

    @abstractmethod
    def load_json(self, path): pass

    @abstractmethod
    def write_log(self): pass
```

`YamlConfigParser` and `JsonConfigParser` need to use `ConfigParser` interface, but only need `load_yaml()` and `load_json()` respectively. Problems aries:

- `YamlConfigParser` and `JsonConfigParser` depend on more than it needs -- that would violate ISP.

with ISP

Method 1

After following ISP, splitting the interface into smaller pieces:

```python
from abc import ABC, abstractmethod

class YamlReader(ABC):
    @abstractmethod
    def load(self, path: str) -> dict:
        pass

class JsonReader(ABC):
    @abstractmethod
    def load(self, path: str) -> dict:
        pass
```

This is a case of **over-segregation** -- meaning splitting the interface too much, making it more complex than needed, and losing polymorphism. The above method (`ConfigParser` with `load()`) is more suitable in this case since each parser only implements what it needs (`load()`).

Method 2

If `YamlConfigParser` and `JsonConfigParser` are as follows:

```python
class YamlConfigParser(YamlReader):
    def read_yaml(self, path):
        with open(path, "r") as file:
            return yaml.safe_load(file)


class JsonConfigParser(JsonReader):
    def read_json(self, path):
        with open(path, "r") as file:
            return json.load(file)
```

Now it is a valid use of the ISP. To demonstrate the ISP effect, the configuration of `YamlConfigParser` and `JsonConfigParser` is modified as mentioned above to validate the ISP.