Dependency Inversion Principle (DIP)

> High-level modules should not depend on low-level modules. Both must depend on abstractions. Abstractions should not depend on details. Details should depend on abstractions.

without DIP

```python
class ConfigLoader:
    def __init__(self):
        self.parser = YamlConfigParser()  # tightly bound to YAML

    def load(self, path):
        return self.parser.read_yaml(path)
```

It is an example of tight coupling. Disadvantages are:

- Tightly coupled to `YamlConfigParser`.
- If later want to support JSON, TOML, etc., it must be modified.

This is **inversion gone wrong** — the high-level logic (`ConfigLoader`) depends directly on low-level details (`YamlConfigParser`).

with DIP

DIP suggests flip the dependencies:

- `ConfigLoader` should depend on an abstract interface like `ConfigReader`
- Then inject the concrete parser (e.g., `YamlConfigParser`) from the outside