Python Open–closed principle

Summary: in this tutorial, you'll learn about the open-closed principle to extend the system without directly modifying existing code.

Introduction to the open-closed principle

The open-closed principle is one of the five principles in the SOLID principle. The letter O in the SOLID stands for the open-closed principle.

- **S** – Single Responsibility Principle (https://www.pythontutorial.net/python-oop/python-single-responsibility-principle/)
- **O** – Open-closed Principle
- **L** – Liskov Substitution Principle (https://www.pythontutorial.net/python-oop/python-liskov-substitution-principle/)
- **I** – Interface Segregation Principle (https://www.pythontutorial.net/python-oop/python-interface-segregation-principle/)
- **D** – Dependency Inversion Principle (https://www.pythontutorial.net/python-oop/python-dependency-inversion-principle/)

The open-closed principle states that a class, (https://www.pythontutorial.net/python-oop/python-class/) method (https://www.pythontutorial.net/python-oop/python-methods/) , and function (https://www.pythontutorial.net/python-basics/python-functions/) should be open for extension but closed for modification.

The open-closed principle sounds contradictory.

The purpose of the open-closed principle is to make it easy to add new features (or use cases) to the system without directly modifying the existing code.

Consider the following example:

```
class Person:
    def __init__(self, name):
        self.name = name

    def __repr__(self):
        return f'Person(name={self.name})'
```

```
class PersonStorage:
    def save_to_database(self, person):
        print(f'Save the {person} to database')

    def save_to_json(self, person):
        print(f'Save the {person} to a JSON file')


if __name__ == '__main__':
    person = Person('John Doe')
    storage = PersonStorage()
    storage.save_to_database(person)
```

In this example, the `PersonStorage` class has two methods:

- The save_to_database() method saves a person to the database.
- The save_to_json() method saves a person to a JSON file.

Later, if you want to save the Person's object into an XML file, you must modify the `PersonStorage` class. It means that the PersonStorage class is not open for extension but modification. Hence, it violates the open-closed principle.

The open-closed principle example

To make the `PersonStorage` class conforms with the open-closed principle; you need to design the classes so that when you need to save the Person's object into a different file format, you don't need to modify it.

See the following class diagram:

First, define the `PersonStorage` abstract class (https://www.pythontutorial.net/python-oop/python-abstract-class/) that contains the `save()` abstract method:

```
from abc import ABC, abstractmethod

class PersonStorage(ABC):
    @abstractmethod
    def save(self, person):
        pass
```

Second, create two classes `PersonDB` and `PersonJSON` that save the `Person` object into the database and JSON file. These classes inherit from the `PersonStorage` class:

```
class PersonDB(PersonStorage):
    def save(self, person):
        print(f'Save the {person} to database')
```

```
class PersonJSON(PersonStorage):
    def save(self, person):
        print(f'Save the {person} to a JSON file')
```

To save the `Person` object into an XML file, you can define a new class `PersonXML` that inherits from the `PersonStorage` class like this:

```
class PersonXML(PersonStorage):
    def save(self, person):
        print(f'Save the {person} to a JSON file')
```

And you can save the `Person` 's object into an XML file using the `PersonXML` class:

```
if __name__ == '__main__':
    person = Person('John Doe')
    storage = PersonXML()
    storage.save(person)
```

Put it all together:

```
from abc import ABC, abstractmethod


class Person:
    def __init__(self, name):
        self.name = name

    def __repr__(self):
        return f'Person(name={self.name})'


class PersonStorage(ABC):
    @abstractmethod
    def save(self, person):
        pass


class PersonDB(PersonStorage):
    def save(self, person):
        print(f'Save the {person} to database')


class PersonJSON(PersonStorage):
    def save(self, person):
        print(f'Save the {person} to a JSON file')
```

```
class PersonXML(PersonStorage):
    def save(self, person):
        print(f'Save the {person} to a XML file')


if __name__ == '__main__':
    person = Person('John Doe')
    storage = PersonXML()
    storage.save(person)
```

Summary

- The open-closed principle allows you to design the system so that it is open for extension but closed for modification.