Open-Closed Principle

Ein ausgewähltes SOLID-Prinzip

Agenda

1. Definition
2. Motivation
3. Implementierungsmöglichkeiten
4. UML-Diagramm
5. Demo

Ivar Jacobson:

“All systems change during their life cycles. This must be borne in mind when developing systems expected to last longer than the first version.”

(Object Oriented Software Engineering a Use Case Driven Approach, Ivar Jacobson, Addison Wesley, 1992, p 21.)

How can we create designs that are stable in the face of change and that will last longer than the first version?

Definition

Das **Open-Closed-Prinzip** besagt, dass Software-Einheiten offen für Erweiterungen, gleichzeitig aber geschlossen gegenüber Veränderungen sein sollen.

Bertrand Meyer (1988):
“Modules should be both open (for extension) and closed (for modification).”
(https://de.wikipedia.org/wiki/Prinzipien_objektorientierten_Designs#Open-Closed-Prinzip)

Motivation

Neue Funktionen einem System hinzufügen, ohne das bestehende System im Kern zu manipulieren.

- ➔ **Geringerer Implementierungsaufwand**
  - Nur die eigene Implementierung muss bewältigt werden
- ➔ **Reduzierung des Fehlerpotentials**
  - Es werden an weniger Programmteilen Änderungen vorgenommen
  - Man kann sich auf die Funktionalität vorhandener Implementierungen weiterhin verlassen (auch andere davon abhängige Implementierungen)
- ➔ **Zusammenfassend: Weniger aufwändige Erweiterung bestehender Implementierungen**

Implementierungsmöglichkeiten

Allgemein: Hierarchie von Implementierungen von sehr abstrakt zu immer spezifischer

- **Vererbung**
  - Von der Basisklasse werden Subklassen abgeleitet, die die vorhandene Funktionalität erweitern, ohne das die Basisklasse angerührt werden muss
- **Interfaces**
  - Allgemeine Implementierung, was eine Funktion tun soll, funktionale Implementierung an notwendiger Stelle
- **Duck-Typing**
  - “When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck” - James Whitcomb Rileys

UML Diagramm

```
1   class Animal:
2       def __init__(self, name):
3           self.name = name
4
5       def get_name(self):
6           return self.name
7
8   animals = [
9       Animal('lion'),
10      Animal('mouse')
11  ]
12
13  def animal_sound(animals: list):
14      for animal in animals:
15          if animal.name == 'lion':
16              print('roar')
17
18          elif animal.name == 'mouse':
19              print('squeak')
20
21  animal_sound(animals)
```

UML Diagramm

```
1   class Animal:
2       def __init__(self, name):
3           self.name = name
4
5       def get_name(self):
6           return self.name
7
8   animals = [
9       Animal('lion'),
10      Animal('mouse')
11  ]
12
13  def animal_sound(animals: list):
14      for animal in animals:
15          if animal.name == 'lion':
16              print('roar')
17
18          elif animal.name == 'mouse':
19              print('squeak')
20
21  animal_sound(animals)
```

```
Animal('snake')
```

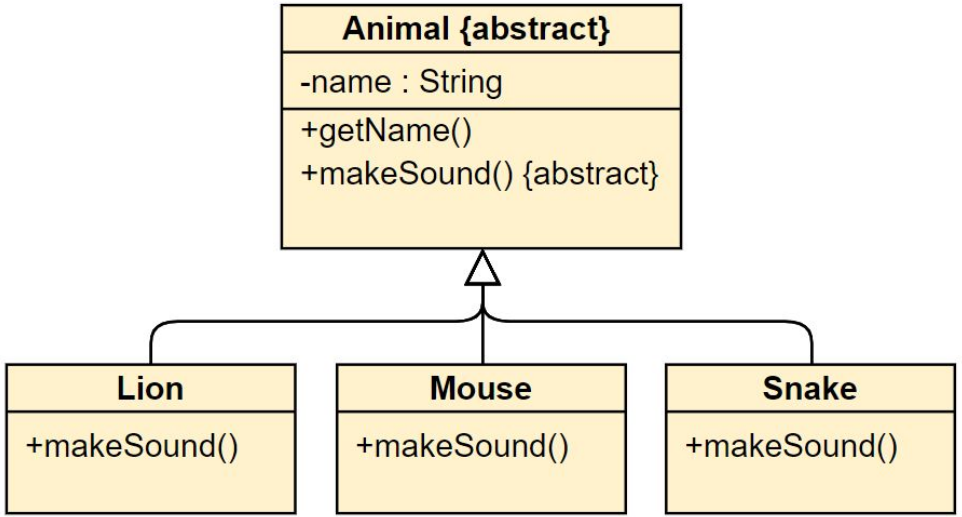
```
elif animal.name == 'snake':
    print('hiss')
```

UML Diagramm

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

    def get_name(self):
        return self.name

animals = [
    Animal('lion'),
    Animal('mouse')
]

def animal_sound(animals: list):
    for animal in animals:
        if animal.name == 'lion':
            print('roar')

        elif animal.name == 'mouse':
            print('squeak')

animal_sound(animals)
```

```python
Animal('snake')
```

```python
elif animal.name == 'snake':
    print('hiss')
```



Code Beispiel

Quellen

- https://de.wikipedia.org/wiki/Ivar_Jacobson
- https://www.oose.de/blogpost/30-jahre-anwendungsfaelle/
- Object Oriented Software Engineering a Use Case Driven Approach, Ivar Jacobson, Addison Wesley, 1992, p 21.
- https://drive.google.com/file/d/0BwhCYaYDn8EgN2M5MTkwM2EtNWFkZC00ZTI3LWFjZTUtNTFhZGZiYmUzOD1view
- https://de.wikipedia.org/wiki/Prinzipien_objektorientierten_Designs#Open-Closed-Prinzip
- https://de.wikipedia.org/wiki/Bertrand_Meyer