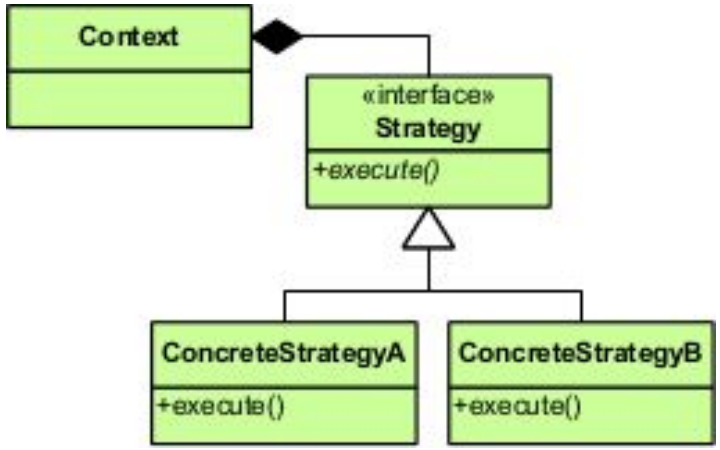
Strategy Pattern

Strategy Pattern

El patrón **Estrategia** (Strategy) es un patrón de diseño para el desarrollo de software. Se clasifica como patrón de comportamiento porque determina cómo se debe realizar el intercambio de mensajes entre diferentes objetos para resolver una tarea. El patrón estrategia permite mantener un conjunto de algoritmos de entre los cuales el objeto cliente puede elegir aquel que le conviene e intercambiarlo dinámicamente según sus necesidades.



Strategy Pattern - Con Clases

```python
import abc


class Strategy(object):

    __metaclass__ = abc.ABCMeta

    @abc.abstractmethod
    def execute(self): pass


class Strategy1(Strategy):

    def execute(self): print("Strategy 1")


class Strategy2(Strategy):

    def execute(self): print("Strategy 2")


# ================================================================
class Context(object):

    def __init__(self, strategy):
        assert isinstantce(strategy, Strategy)
        self.strategy = strategy

    def execute(self):
        return self.strategy.execute()
```

Strategy Pattern - Con Clases

```
import abc


class Strategy(object):

    __metaclass__ = abc.ABCMeta
```

```
>>> ctx1 = Context(Strategy1())
>>> ctx2 = Contect(Strategy2())

>>> ctx1.execute()
Strategy 1
>>> ctx2.execute()
Strategy 2
```

```
class Strategy2(Strategy):

    def execute(self): print("Strategy 2")


# ============================================================================
class Context(object):

    def __init__(self, strategy):
        assert isinstantce(strategy, Strategy)
        self.strategy = strategy

    def execute(self):
        return self.strategy.execute()
```

Strategy Pattern - Con Clases

```
import abc


class Strategy(object):

    __metaclass__ = abc.ABCMeta
```

```
>>> ctx1 = Context(Strategy1())
>>> ctx2 = Contect(Strategy2())

>>> ctx1.execute()
Strategy 1
>>> ctx2.execute()
Strategy 2
```

```
class Strategy2(Strategy):

    def execute(self): print("Strategy 2")


# ============================================================================
class Context(object):

    def __init__(self, strategy):
        assert isinstantce(strategy, Strategy)
        self.strategy = strategy

    def execute(self):
        return self.strategy.execute()
```

Strategy Pattern - Con Funciones

```
def strategy1(): print("Strategy 1")


def strategy2(): print("Strategy 2")


# ============================================================================
class Context(object):

    def __init__(self, strategy):
        assert hasattr(strategy, "__call__")
        self.strategy = strategy

    def execute(self):
        return self.strategy()



>>> ctx1 = Context(strategy1)
>>> ctx2 = Contect(strategy2)

>>> ctx1.execute()
Strategy 1
>>> ctx2.execute()
Strategy 2
```

Strategy Pattern - Con Funciones 2

```
class Context(object):

    def __init__(self, strategy):
        if strategy is not None:
            assert hasattr(strategy, "__call__")
        self.execute = strategy

    def execute(self):
        return "Default Behavior"



>>> ctx1 = Context(lambda: "Strategy 1")
>>> ctx2 = Context()

>>> ctx1.execute()
"Strategy 1"
>>> ctx2.execute()
"Default Behavior"
```

Strategy Pattern - Ejemplo

```
class Context(object):

    def __init__(self, strategy=None):
        if strategy is not None:
            assert hasattr(strategy, "__call__")
        self.execute = strategy

    def execute(self, num0, num1):
        return num0 + num1



>>> ctx1 = Context()
>>> ctx2 = Context(pow)

>>> ctx1.execute(2, 3)
5
>>> ctx2.execute(2, 3)
8
```