Design Patterns

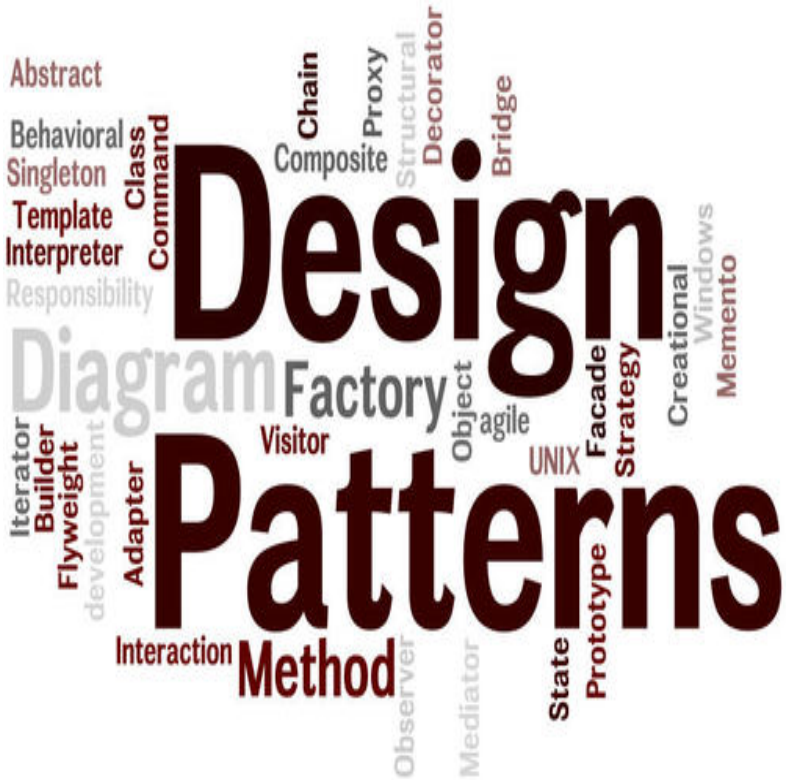
Trần Hữu Thúy

Ngày 4 tháng 3 năm 2020

Nội dung

Tổng quan về Design Patterns

- Code cần chứa tất cả lợi ích được cung cấp bởi ngôn ngữ hướng đối tượng.
- Đoạn code cần có tính linh hoạt và bất kì sự chỉnh sửa nào đều là rất ít.
- Design Patterns là kinh nghiệm trong thiết kế các đoạn mã hướng đối tượng.
- Patterns không phải là code hoàn chỉnh nhưng nó có thê sử dụng như một template để giải quyết bài toán.

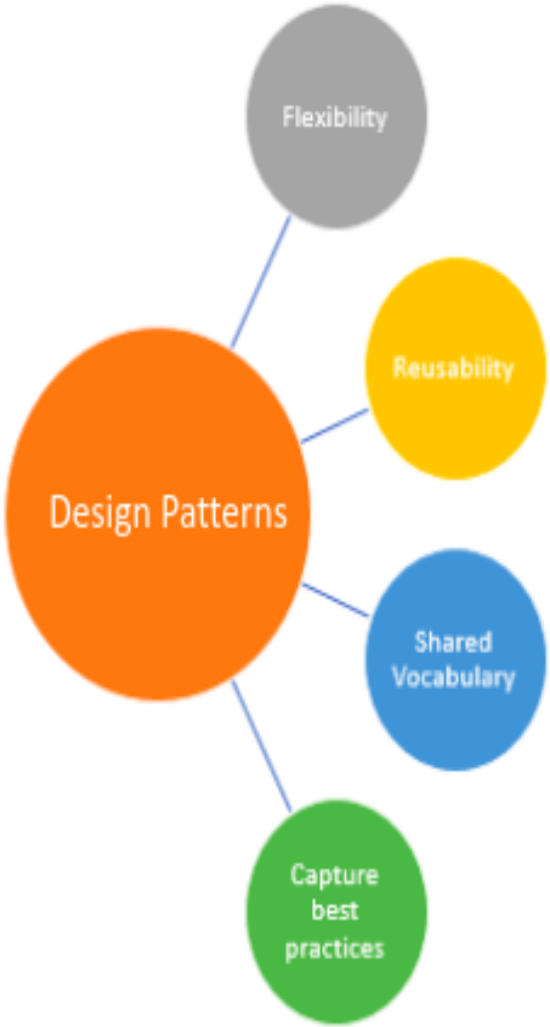


Tổng quan về Design

Thành phần chính của một Patterns

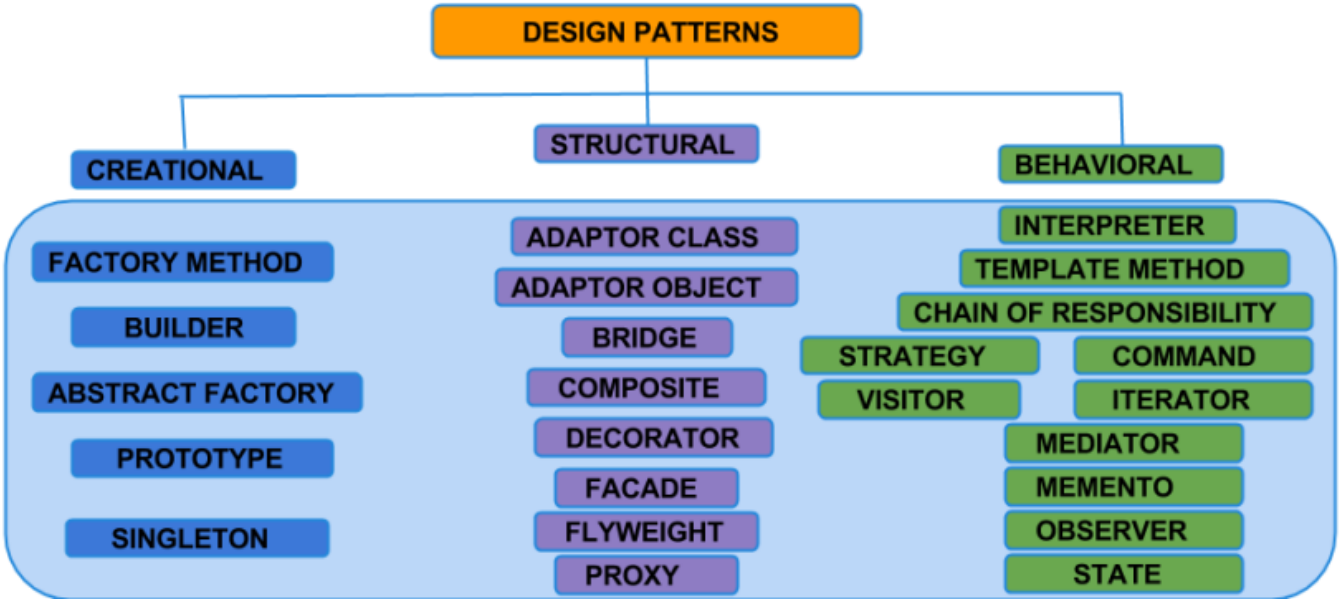
- Patterns name : tên của pattern, giúp hình dung, định nghĩa bài toán, giải pháp của nó.
- The problem describes when to apply the pattern : Giải thích bài toán và ngữ cảnh sử dụng nó.
- Solution : Mô tả các thành phần, mối quan hệ,...Một solution không phải code đầy đủ, nó như một bản mẫu cần điền code còn thiếu.
- Results and consequences : Kết quả , hậu quả, ý nghĩa mà nó đem lại cho việc thiết kế.

Đặc điểm của Design Patterns



- Design Patterns giúp code trở nên linh hoạt và có thể tái sử dụng.
- Việc code trở nên đơn gian, dễ dàng chia sẻ với lập trình viên khác.

Phân loại Design Patterns



- Creational : Design Patterns này được sử dụng để khởi tạo đối tượng.
- Structural : Tổ chức các đối tượng, lớp để thành những cấu trúc lớn hơn.
- Behavioral : Xác định cách giao tiếp giữa các đối tượng.

Singleton Design Pattern

- Singleton đảm bảo chỉ có một đội tượng lớp được tạo ra.
- Singleton thuộc nhóm Cretional Design Patterns.

Singleton Design Pattern

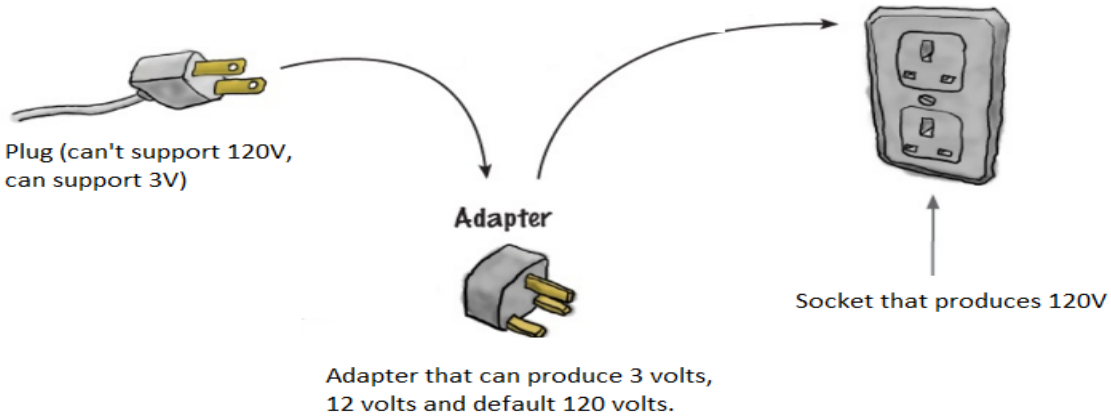
- Đối tượng được tạo ra duy nhất là biến static sc.Trong lần đầu tiên gọi phương thức getInstance, sc được khởi tạo.
- Các lần gọi getIntance sau, sẽ luôn trả về biến toàn cục sc.

```
thuy@Tran_Huu_Thuy:~/Desktop$ python3 hello.py
First  <__main__.Singleton object at 0x7f3b1278de48>
Second  <__main__.Singleton object at 0x7f3b1278de48>
Third  <__main__.Singleton object at 0x7f3b1278de48>
thuy@Tran_Huu_Thuy:~/Desktop$
```

Open

```
class Singleton():

    # static property
    _sc = None

    def __init__(self):
        if Singleton._sc is None:
            Singleton._sc = self
        else:
            print('This is Singleton Class')

    @staticmethod
    def getInstance():
        if Singleton._sc is None:
            return Singleton()
        return Singleton._sc

print('First ', Singleton.getInstance())
print('Second ', Singleton.getInstance())
print('Third ', Singleton.getInstance())
```

Adapter Design Pattern

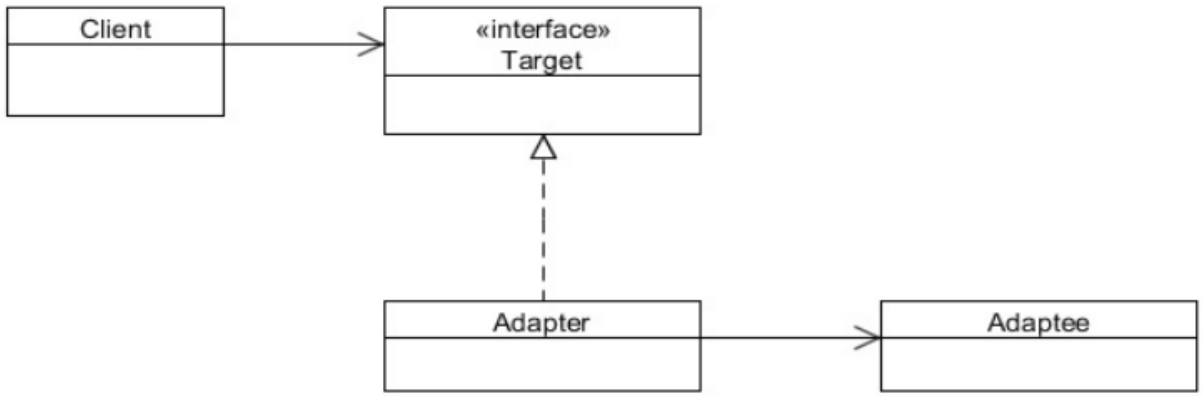


- Adapter Design Patterns giống như khi ta không thể kết nối trực tiếp hai thiết mà phải thông qua thiết bị trung gian.
- Nhờ có thiết bị trung gian này mà ta có thể dễ dàng thay đôi một trong hai thiết bị mà adapter vẫn đáp ứng được.

Adapter Design Pattern

Sơ đồ

Hình: Biểu đồ UML



- Sơ đồ cho thấy việc kết nối Client và Adaptee cần thông qua một Adapter.
- Lớp Adapter thường sẽ là interface.Adaptee cũng vậy để dễ dàng thay đổi nó.
- Lớp Client sẽ sử dụng target interface làm thuộc tính, Adapter cũng sẽ sử dụng Adaptee làm thuộc tính.

Adapter Design Pattern

Example

```
# Target Interface
class DatabaseInterface:

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

    def connect(self):
        pass

    def close(self):
        pass

    def queries(self, sql):
        pass
```

- target interface với 3 phương thức chưa được cài đặt.
- Các lớp Adapter kế thừa nó sẽ được lớp client sử dụng.

Adapter Design Pattern

Example(tiếp)

```
# Adaptee Interface
class SqlInterface:
    def __init__(self, name, username, password, host):
        self.name = name
        self.username = username
        self.password = password
        self.host = host
        print('Init ', name, ' database')

    def connect(self):
        pass

    def queries(self, sql='select * from table'):
        pass

    def close(self):
        pass
```

- Các adaptee sẽ kế thừa interface này, để thay đôi sql(mysql -> sql server), ta chỉ cần tạo thêm lớp kế thừa nó.

Adapter Design Pattern

Example(tiếp)

- Tạo Adapter, adpatee từ các interface.

```
class SqlAdapter(DatabaseInterface):
    _sql = None

    def __init__(self, sql):
        self._sql = sql

    def setSql(self, sql):
        self._sql = sql

    def connect(self):
        self._sql.connect()

    def close(self):
        self._sql.close()

    def queries(self, sql):
        self._sql.queries(sql)
```

```
class Mysql(SqlInterface):

    def connect(self):
        print('User :', self.username, ',pass :',
              self.password, ', host :' + self.host)

    def queries(self, sql='select * from table'):
        print('Mysql queries : ' + sql)

    def close(self):
        print('Mysql close')
```

Adapter Design Pattern

Example(tiếp)

- Cuối cùng tạo lớp client và sử dụng lớp trên

```
class Client:
    _adapter = None

    def __init__(self, adapter):
        self._adapter = adapter


    def connect_database(self):
        self._adapter.connect()


mysql = Mysql('mysql', 'root', 'tranhuuthuy', 'localhost')
adapter = SqlAdapter(mysql)

client = Client(adapter)
client.connect_database()
```

Factory Design Patterns

- Factory thuộc nhóm Cretional Design Patterns, được sử dụng trong khi super class có nhiều sub-class và phải trả về một trong số chúng.
- Subclass sẽ được định nghĩa bằng string trong phương thức trả về đối tượng.

Factory Design Patterns

Example

- Parent Class

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

    def information(self):
        pass
```

Factory Design Patterns

Example

- SubClass

```
# Subclass 1
class Samsung(Phone):
    def information(self):
        print('SamSung ', self.name, ' : ', self.cost, ' $')


# Subclass 2
class Iphone(Phone):
    def information(self):
        print('Iphone ', self.name, ' : ', self.cost, ' $')
```

Factory Design Patterns

Example

- Factory, kiểu đối tượng được tuyền vào với string.

```
class PhoneFactory:
    def factoryMethod(self, type, name, cost):
        type_class = str(type).capitalize()
        return globals()[type_class](name, cost)

factory = PhoneFactory()

# samsung
s = factory.factoryMethod('samsung', 'oppo', '1.200')
s.information()
# iphone
i = factory.factoryMethod('iphone', '10', '3.000')
i.information()
```

End