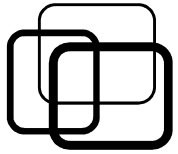
Mẫu Strategy

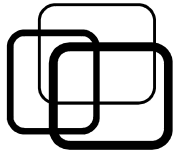
GV. Nguyễn Minh Huy

Mẫu Strategy



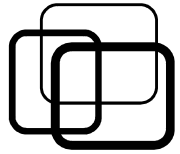
- Ngữ cảnh:
  - Bài toán:
    - Sắp xếp mảng.
    - Thanh toán hóa đơn.
    - Vẽ một loại hình.
  - Mục tiêu:
    - Có nhiều thuật toán xử lý cho một phương thức.
    - Có thể lựa chọn thuật toán khi chương trình thực hiện.
    - Có thể thay đổi thuật toán lựa chọn nhiều lần.

Mẫu Strategy

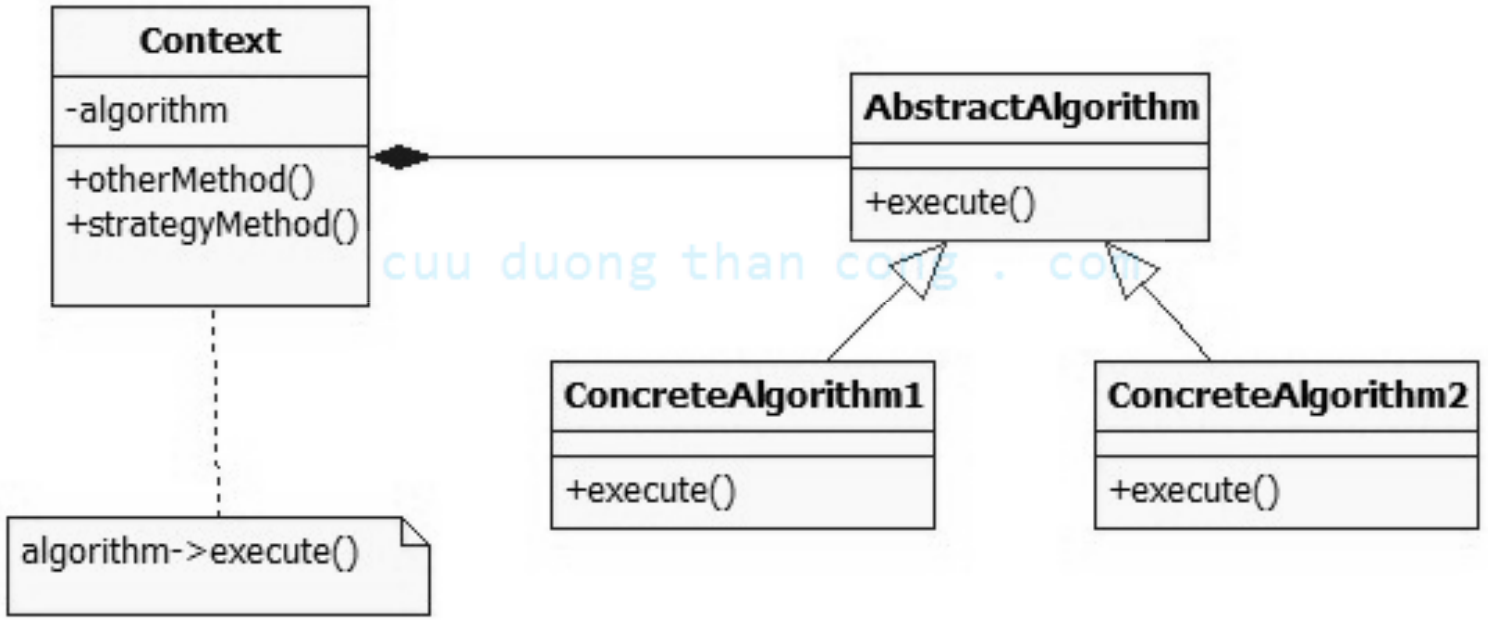


- Hướng tiếp cận:
  - Nhiều thuật toán xử lý khác nhau:
    - Tạo lớp thuật toán tổng quát.
    - Tạo lớp kế thừa cho mỗi thuật toán cụ thể.
  - Lựa chọn thuật toán xử lý cho phương thức:
    - Trang bị đối tượng thuật toán cho lớp chứa phương thức.
    - Khi gọi phương thức, nhờ đối tượng thuật toán thực hiện.
  - Thay đổi thuật toán lựa chọn nhiều lần:
    - Cho phép cập nhật đối tượng thuật toán đã trang bị.

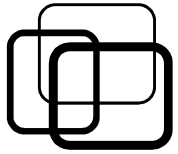
Mẫu Strategy



- Hướng tiếp cận:



Mẫu Strategy

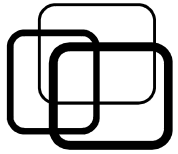


- Cài đặt:

```
class Receipt
{
private:
        Payment* m_pay;
public:
        double calculate();
};


double Receipt::calculate()
{
        return m_pay->calculate();
}
```

```
class Payment
{
public:
        virtual double calculate() = 0;
};

class CashPayment: public Payment
{
public:
        double calculate();
};

class CreditCardPayment: public Payment
{
public:
        double calculate();
};
```

Mẫu Strategy



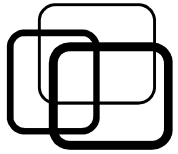
- Các vấn đề xung quanh:
  - Cập nhật đối tượng thuật toán bằng cách nào?
    - Khởi tạo thông qua constructor.
      - ➔ Chỉ lựa chọn 1 lần, không thể thay đổi lần sau.
    - Tạo phương thức truy xuất (get/set).
      - ➔ Phải lưu đối tượng thuật toán.
    - Cập nhật thông qua tham số phương thức.

```
class Receipt
{
private:
    Payment* m_pay;
public:
    Receipt(Payment* pay);
    double calculate();
};
```

```
class Receipt
{
private:
    Payment* m_pay;
public:
    setPay(Payment* pay);
    double calculate();
};
```

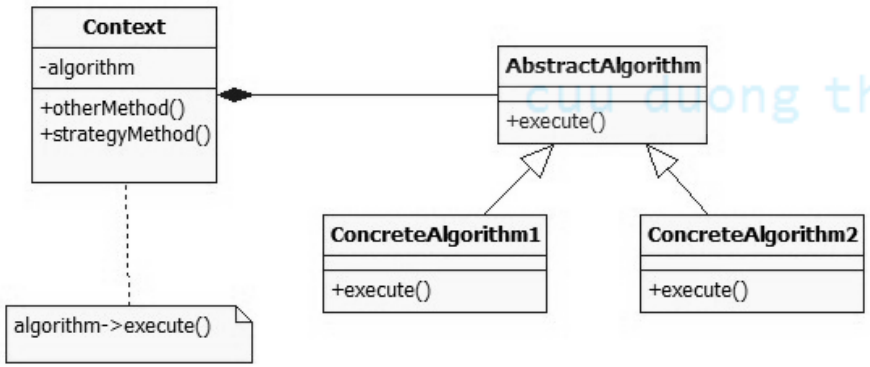
```
class Receipt
{
public:
    double calculate(Payment* pay);
};
```

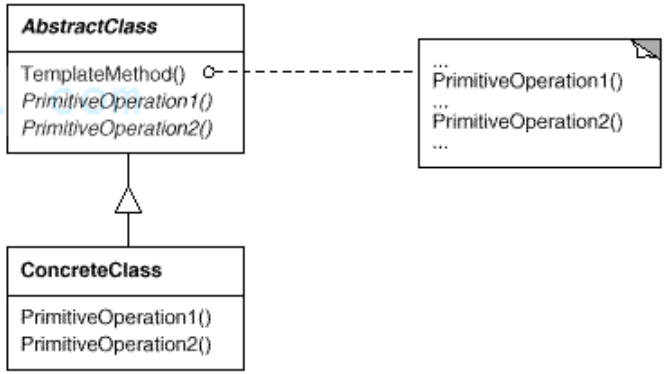
Mẫu Strategy



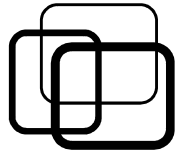
- Các vấn đề xung quanh:
  - Phân biệt Strategy và Template Method:

Tiêu chí	Strategy	Template Method
Ý tưởng	- Thay đổi toàn bộ thuật toán xử lý phương thức. - "Thay da đổi thịt".	- Thay đổi một vài bước trong thuật toán xử lý phương thức. - Giữ nguyên khung sườn.
Cài đặt	- Tách biệt cây thuật toán. - Sử dụng composition đối tượng thuật toán.	- Sử dụng inheritance trên chính lớp chứa phương thức.





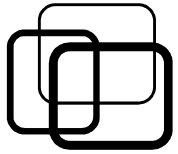
Mẫu Strategy



- Các vấn đề xung quanh:
  - Phân biệt Strategy và kế thừa trực tiếp:
    - Giống nhau: đều có thể thay đổi xử lý của phương thức.

Tiêu chí	**Strategy**	**Kế thừa trực tiếp**
Tính chất	- Quan hệ HAS-A (giữa Context và Algorithm)	- Quan hệ IS-A (giữa Context và Algorithm)
Cài đặt	- Tách biệt cây thuật toán.	- Override lại phương thức. → "Bùng nổ" lớp kế thừa.
Linh động	- Lựa chọn thuật toán động (khi nào cần). - Thay đổi lựa chọn nhiều lần.	- Lựa chọn thuật toán cứng (khi tạo lớp Context).

Mẫu Strategy

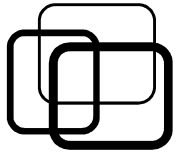


- Các vấn đề xung quanh:
  - Hạn chế:
    - Tách biệt cây thuật toán.
      - ➔ Không thể truy xuất thành phần private lớp Context.

```
class Receipt
{
private:
        Payment* m_pay;
        double m_tax;
public:
        double calculate();
};
double Receipt::calculate()
{
        return m_pay->calculate();
}
```

```
class CashPayment: public Payment
{
public:
        double calculate();
};

double CashPayment::calculate()
{
        // Làm sao truy xuất m_tax??
}
```

Mẫu Strategy



- Các vấn đề xung quanh:
  - Hạn chế:
    - Cho phép thay đổi thuật toán xử lý.
      - ➔ Tạo nhiều đối tượng thuật toán.
      - ➔ Tốn tài nguyên hệ thống.
      - ➔ Áp dụng Singleton!!