

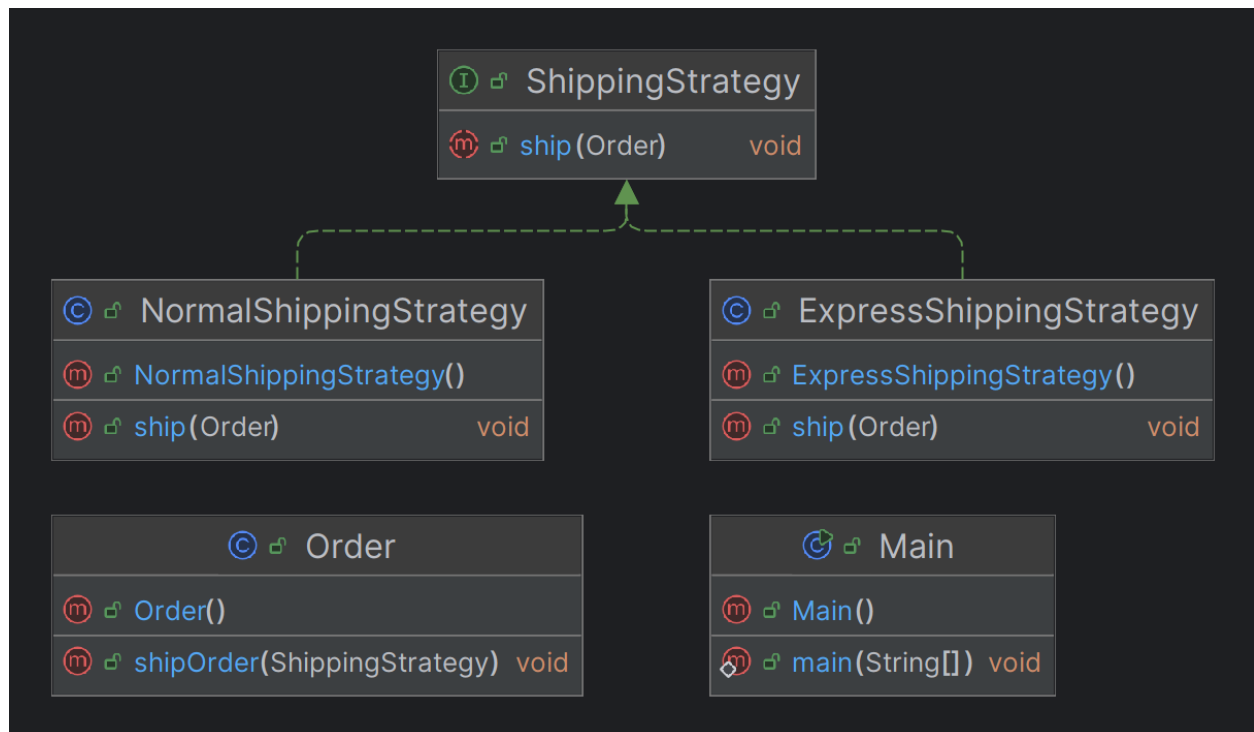
Họ và tên: Nguyễn Khánh Quy

MSSV: 21110282

Strategy Pattern

Mẫu **Strategy** trong lập trình hướng đối tượng là một mẫu thiết kế hành vi được sử dụng để tách biệt và thay đổi linh hoạt các thuật toán, chiến lược trong một hệ thống mà không làm ảnh hưởng đến các đối tượng sử dụng chúng. **Strategy** giúp cho việc thay đổi hoặc mở rộng các chiến lược trở nên dễ dàng hơn mà không cần sửa đổi nhiều mã nguồn.

Dưới đây là một ví dụ minh họa về mẫu **Strategy** được triển khai bằng ngôn ngữ lập trình Java. Trong ví dụ này, chúng ta giả định một hệ thống quản lý đơn hàng cần có khả năng vận chuyển các đơn hàng bằng nhiều phương thức khác nhau, như vận chuyển thông thường và vận chuyển nhanh. Mẫu **Strategy** được áp dụng để tách biệt logic vận chuyển ra khỏi đơn hàng và cho phép dễ dàng thêm mới và thay đổi các phương thức vận chuyển mà không ảnh hưởng đến các đối tượng đơn hàng.



Bước 1: Xác định Interface (ShippingStrategy):

Đầu tiên, chúng ta cần tạo một interface ShippingStrategy. Interface này định nghĩa một hành vi chung cho các chiến lược vận chuyển.

```
public interface ShippingStrategy {  
    void ship(Order order);  
}
```

Bước 2: Tạo các lớp (NormalShippingStrategy, ExpressShippingStrategy):

Chúng ta tạo các lớp NormalShippingStrategy và ExpressShippingStrategy để cung cấp cài đặt cụ thể cho phương thức vận chuyển. Mỗi lớp này triển khai phương thức ship() từ interface ShippingStrategy.

```
public class NormalShippingStrategy implements ShippingStrategy {  
    @Override  
    public void ship(Order order) {  
        System.out.println("Using normal shipping.");  
    }  
}
```

```
public class ExpressShippingStrategy implements ShippingStrategy {  
    @Override  
    public void ship(Order order) {  
        System.out.println("Using express shipping.");  
    }  
}
```

Bước 3: Tạo lớp (Order):

Trong lớp Order, chúng ta có một phương thức shipOrder(ShippingStrategy shippingStrategy) để vận chuyển đơn hàng. Phương thức này nhận một đối tượng ShippingStrategy và gọi phương thức ship() của nó.

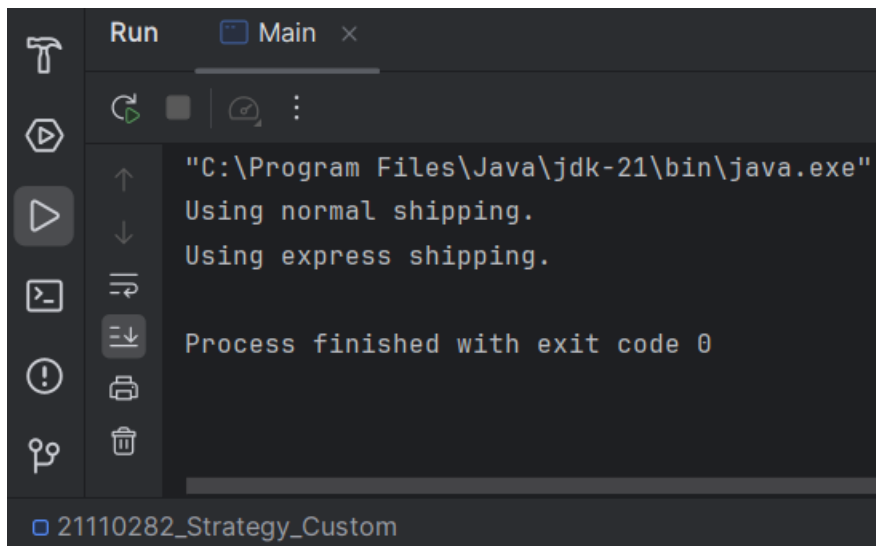
```
public class Order {  
  
    public void shipOrder(ShippingStrategy shippingStrategy) {  
        if (shippingStrategy != null) {  
            shippingStrategy.ship(this);  
        } else {  
            System.out.println("No shipping strategy provided.");  
        }  
    }  
}
```

Bước 4: Thử nghiệm

Tạo một lớp Main để thử nghiệm, chúng ta tạo một đơn hàng và sử dụng các chiến lược vận chuyển khác nhau để vận chuyển đơn hàng đó.

```
public class Main {  
  
    public static void main(String[] args) {  
        // Tạo đơn hàng  
        Order order = new Order();  
  
        // Vận chuyển bằng phương thức vận chuyển Normal Shipping  
        ShippingStrategy normalShippingStrategy = new  
NormalShippingStrategy();  
        order.shipOrder(normalShippingStrategy);  
  
        // Vận chuyển bằng phương thức vận chuyển Express Shipping  
        ShippingStrategy expressShippingStrategy = new  
ExpressShippingStrategy();  
        order.shipOrder(expressShippingStrategy);  
    }  
}
```

Kết quả:



```
Run Main x
"C:\Program Files\Java\jdk-21\bin\java.exe"
Using normal shipping.
Using express shipping.
Process finished with exit code 0
21110282_Strategy_Custom
```

Trong ví dụ này, Order cho phép thay đổi phương thức vận chuyển tùy ý mà không cần thay đổi mã nguồn của nó. Điều này làm cho mã nguồn linh hoạt hơn và dễ bảo trì, phát triển trong tương lai.