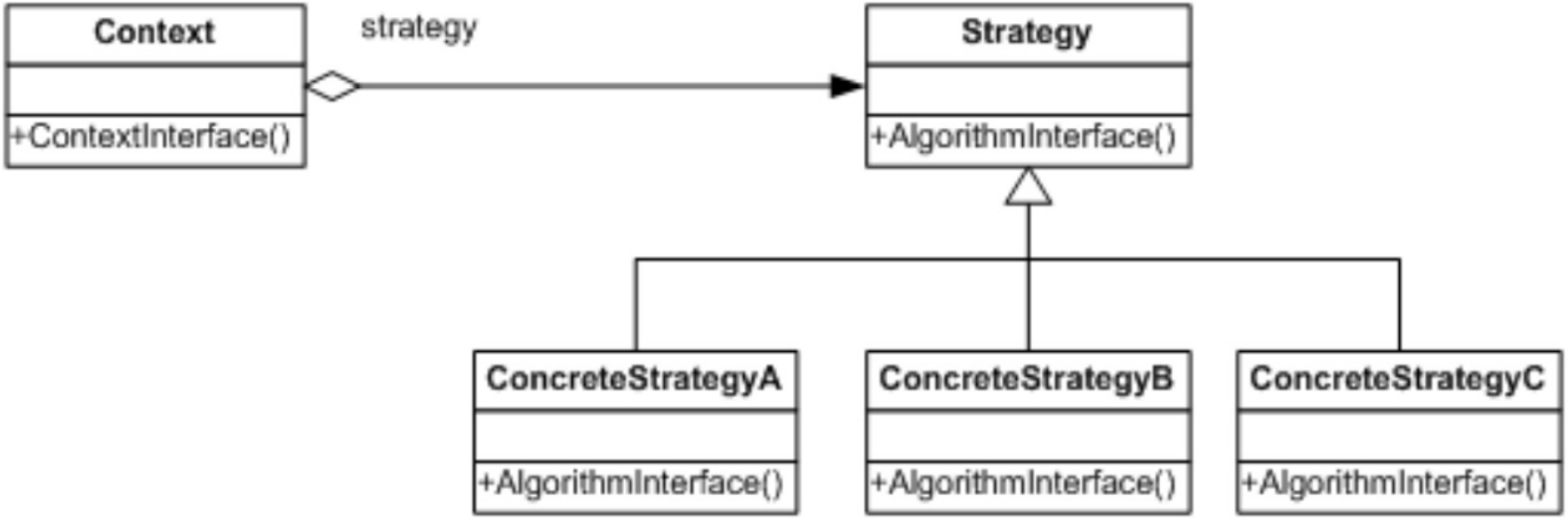
Strategy

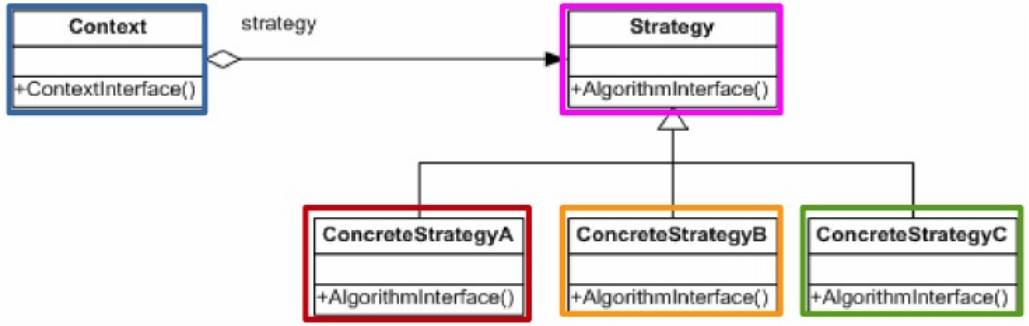
Padrão de Projeto *Strategy*

Descrição do problema

- O padrão *Strategy* é usado quando temos uma família de algoritmos e desejamos encapsular cada um deles e torná-los intercambiáveis.



Exemplo com Strategy




```java
public class Compra {
    private MetodoPagamento metodo;

    public Compra(MetodoPagamento metodo) {
        this.metodo = metodo;
    }

    public double calcularValorCompra(double gasto) {
        return metodo.calcularValor(gasto);
    }
}
```

```java
public interface MetodoPagamento {
    public double calcularValor(double gasto);
}
```

```java
public class Dinheiro implements MetodoPagamento{
    @Override
    public double calcularValor(double gasto) {
        return gasto;
    }
}
```

```java
public class CartaoDebito implements MetodoPagamento{
    @Override
    public double calcularValor(double gasto) {
        return 1.03 * gasto;
    }
}
```

```java
public class CartaoCredito implements MetodoPagamento{
    @Override
    public double calcularValor(double gasto) {
        return 1.1 * gasto;
    }
}
```

Exemplo com Strategy

Strategy (executando)

```
public class TesteCompra {
    public static void main(String[] args) {
        Compra minhaCompra;
        double resultado;

        minhaCompra = new Compra(new Dinheiro());
        resultado = minhaCompra.calcularValorCompra(100.00f);
        System.out.printf("O valor total a ser pago em dinheiro é R$ %.2f\n",resultado);

        minhaCompra = new Compra(new CartaoDebito());
        resultado = minhaCompra.calcularValorCompra(100.00f);
        System.out.printf("O valor total a ser pago no débito é R$ %.2f\n",resultado);

        minhaCompra = new Compra(new CartaoCredito());
        resultado = minhaCompra.calcularValorCompra(100.00f);
        System.out.printf("O valor total a ser pago no crédito é R$ %.2f\n",resultado);
    }
}
```

Utilizamos diferentes algoritmos apenas configurando a estratégia, sem a necessidade de estruturas de seleção.

Exemplo sem Strategy

A classe Compra define vários comportamentos, sendo que eles aparecem como múltiplos comandos condicionais.

```java
public class Compra {
  private String metodo;

  public Compra(String metodo) {
    this.metodo = metodo;
  }

  public double calcularValorCompra(double gasto) {
    double valor = 0;
    if(metodo.equals("Dinheiro")) {
      valor = gasto;
    }
    else if(metodo.equals("CartaoDebito")) {
      valor = 1.03 * gasto;
    }
    else if(metodo.equals("CartaoCredito")) {
      valor = 1.1 * gasto;
    }
    return valor;
  }
}
```

```java
public class TesteCompra {
    public static void main(String[] args) {
        Compra minhaCompra;
        double resultado;

        minhaCompra = new Compra("Dinheiro");
        resultado = minhaCompra.calcularValorCompra(100.00f);
        System.out.printf("0 valor total a ser pago em dinheiro é R$ %.2f\n",resultado);

        minhaCompra = new Compra("CartaoDebito");
        resultado = minhaCompra.calcularValorCompra(100.00f);
        System.out.printf("0 valor total a ser pago no débito é R$ %.2f\n",resultado);

        minhaCompra = new Compra("CartaoCredito");
        resultado = minhaCompra.calcularValorCompra(100.00f);
        System.out.printf("0 valor total a ser pago no crédito é R$ %.2f\n",resultado);
    }
}
```