Strategy

Instituto Federal do Espirito Santo – Campus Colatina
Disciplina: Projeto de Sistemas
Professor: Giovany Frossard Teixeira

Elementos Essenciais

- **Nome:** *Strategy* (também conhecido como *Policy*)
- **Problema:** Existem algoritmos que possuem fluxos de execução de acordo com algumas condições; como por exemplo o tipo do objeto ou outras condições mais específicas. Esses algoritmos podem se tornar grandes e difíceis de fazer manutenção, além de serem difíceis de compreender.
- **Solução:** Definir uma família de algoritmos, encapsular cada uma delas (em classes) e torná-las intercambiáveis.
- **Consequências:** Facilidade de acrescentar novos comportamentos (facilidade de manutenção), possibilidade da implementação ser trocada em tempo de execução (é só trocar a classe que está sendo executada) tornando o comportamento mais dinâmico. Ocorre também o aumento no número de classes criadas (uma para cada fluxo), além da possibilidade de erro (o programador pode esquecer de atribuir a classe concreta que executa o fluxo do algoritmo).

Problema

- Suponha uma empresa, nesta empresa existem um conjunto de cargos, para cada cargo existem regras de cálculo de imposto, determinada porcentagem do salário deve ser retirada de acordo com o salário base do funcionário. Vamos as regras:
  - O Desenvolvedor deve ter um imposto de 15% caso seu salário seja maior que R$ 2000,00 e 10% caso contrário;
  - O Gerente deve ter um imposto de 20% caso seu salário seja maior que R$ 3500,00 e 15% caso contrário;
  - O DBA deve ter um imposto de de 15% caso seu salário seja maior que R$ 2000,00 e 10% caso contrário;

Fonte: https://brizeno.wordpress.com/2011/08/31/strategy/ (acessado em 21/01/2020)

Solução

```java
public double calcularSalarioComImposto() {
        switch (cargo) {
        case DESENVOLVEDOR:
            if (salarioBase >= 2000) {
                return salarioBase * 0.85;
            } else {
                return salarioBase * 0.9;
            }
        case GERENTE:
            if (salarioBase >= 3500) {
                return salarioBase * 0.8;
            } else {
                return salarioBase * 0.85;
            }
        case DBA:
            if (salarioBase >= 2000) {
                return salarioBase * 0.85;
            } else {
                return salarioBase * 0.9;
            }
        default:
            throw new RuntimeException("Cargo não encontrado :/");
        }
    }
```

- O código acima seria implementado na classe Funcionario e possibilitaria tratar o problema descrito no slide anterior
- Essa solução é ruim de fazer manutenção e replica código em diversos locais (veja que o código para DBA e DESENVOLVEDOR é o mesmo).
- Uma solução adequada para esse problema é o uso de Strategy nos fluxos de cada tipo de funcionário.

Solução

```java
package strategy;

interface CalculaImposto
{
        double calculaSalarioComImposto(Funcionario umFuncionario);
}
```

```java
package strategy;

public class CalculoImpostoQuinzeOuDez implements CalculaImposto
{
        @Override
        public double calculaSalarioComImposto(Funcionario umFuncionario)
        {
                if (umFuncionario.getSalarioBase() > 2000)
                {
                        return umFuncionario.getSalarioBase() * 0.85;
                }
                return umFuncionario.getSalarioBase() * 0.9;
        }
}
```

```java
package strategy;

public class CalculoImpostoVinteOuQuinze implements CalculaImposto
{
        @Override
        public double calculaSalarioComImposto(Funcionario umFuncionario)
        {
                if (umFuncionario.getSalarioBase() > 3500)
                {
                        return umFuncionario.getSalarioBase() * 0.8;
                }
                return umFuncionario.getSalarioBase() * 0.85;
        }
}
```

```java
package strategy;
public class Funcionario {
        public static final int DESENVOLVEDOR = 1;
        public static final int GERENTE = 2;
        public static final int DBA = 3;

        protected double salarioBase;
        protected int cargo;
        protected CalculaImposto estrategiaDeCalculo;

        public Funcionario(int cargo, double salarioBase)
        {
            this.salarioBase = salarioBase;
            switch (cargo)
            {
                case DESENVOLVEDOR:
                        estrategiaDeCalculo = new CalculoImpostoQuinzeOuDez();
                        cargo = DESENVOLVEDOR;
                        break;
                case DBA:
                        estrategiaDeCalculo = new CalculoImpostoQuinzeOuDez();
                        cargo = DBA;
                        break;
                case GERENTE:
                        estrategiaDeCalculo = new CalculoImpostoVinteOuQuinze();
                        cargo = GERENTE;
                        break;
                default:
                        throw new RuntimeException("Cargo nao encontrado :/");
            }
        }

        public double calcularSalarioComImposto() {
                return estrategiaDeCalculo.calculaSalarioComImposto(this);
        }

        public double getSalarioBase() {
                return salarioBase;
        }
}
```

Solução

```
package strategy;

public class Strategy
{
    public static void main(String[] args)
    {
                Funcionario umFuncionario = new Funcionario(Funcionario.DESENVOLVEDOR,2100);
                System.out.println(umFuncionario.calcularSalarioComImposto());

                Funcionario outroFuncionario = new Funcionario(Funcionario.DESENVOLVEDOR, 1700);
                System.out.println(outroFuncionario.calcularSalarioComImposto());
    }

}
```

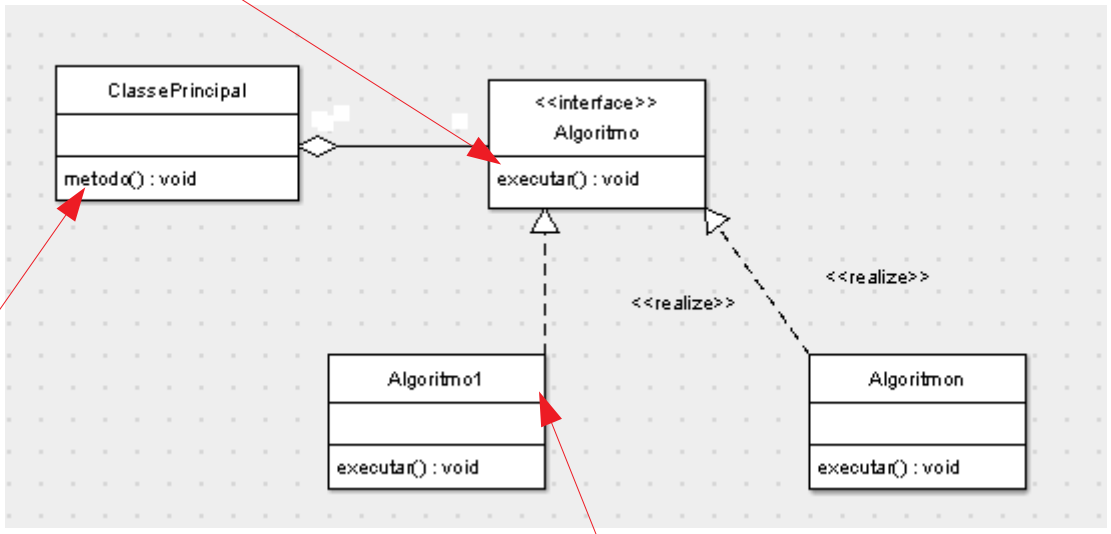
- Note que o uso do *Strategy* também nos obrigou a tratar a questão da criação e configuração do objeto que trata o fluxo de execução.

```
public Funcionario(int cargo, double salarioBase)
{
    this.salarioBase = salarioBase;
    switch (cargo)
    {
        case DESENVOLVEDOR:
                estrategiaDeCalculo = new CalculoImpostoQuinzeOuDez();
                cargo = DESENVOLVEDOR;
                break;
        case DBA:
                estrategiaDeCalculo = new CalculoImpostoQuinzeOuDez();
                cargo = DBA;
                break;
        case GERENTE:
                estrategiaDeCalculo = new CalculoImpostoVinteOuQuinze();
                cargo = GERENTE;
                break;
        default:
                throw new RuntimeException("Cargo nao encontrado :/");
    }
}
```

Esquema visual - UML

```java
package strategy;

interface CalculaImposto
{
        double calculaSalarioComImposto(Funcionario umFuncionario);
}
```




```java
public double calcularSalarioComImposto() {
        return estrategiaDeCalculo.calculaSalarioComImposto(this);
}
```

```java
package strategy;

public class CalculoImpostoQuinzeOuDez implements CalculaImposto
{
        @Override
        public double calculaSalarioComImposto(Funcionario umFuncionario)
        {
                if (umFuncionario.getSalarioBase() > 2000)
                {
                        return umFuncionario.getSalarioBase() * 0.85;
                }
                return umFuncionario.getSalarioBase() * 0.9;
        }
}
```

Já usamos esse padrão sem saber?

- Na disciplina de POO I utilizamos a interface Comparator para a ordenação de listas. A abordagem de criar múltiplos ordenadores (de acordo com o critério que necessitávamos) em muito se assemelha ao padrão *Strategy* e os múltiplos objetos\algoritmos para tratar as possibilidades de diferentes fluxos de execução.
- Na página 37 do Livro Padrões de Projeto – Soluções Reutilizáveis de Software Orientado a Objetos é dito:

  "… Para parametrizar uma rotina de classificação (sort) pelo sistema que ela usa para comparar objetos, nós poderíamos fazer a comparação ...

  2. A responsabilidade de um objeto que é passado para um rotina de classificação (Strategy, 292) ..."
- Entretanto alguns programadores consideram que a interface Comparator utiliza apenas polimorfismo puro (tradicional). Na visão desses falta o **contexto** do padrão *Strategy* que é a multiplicidade de fluxos transformados em objetos para tratar esses fluxos.

Dúvidas?