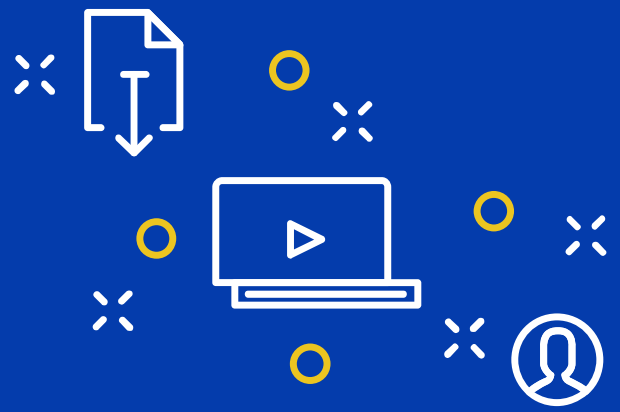
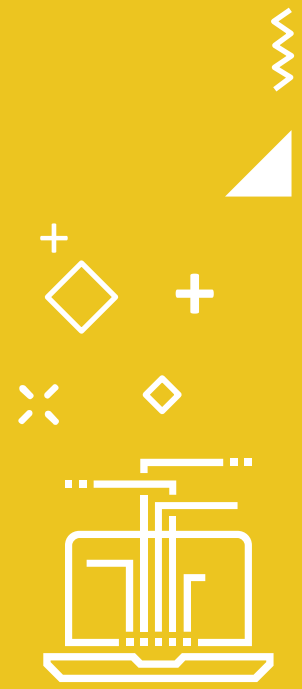
PADRÃO DE PROJETO STRATEGY



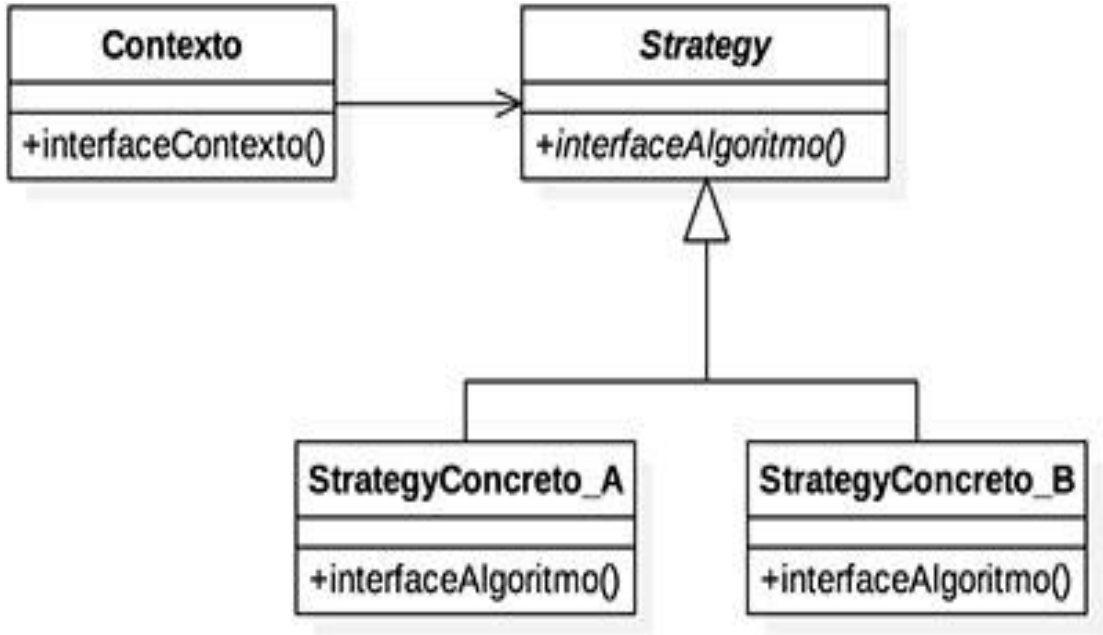
AGENDA



1. *Definição formal do padrão*
2. *Conceitos de Orientação a objetos*
3. *Motivação / Problema*
4. *Discussões sobre o padrão*

Definição Formal

Definir uma família de algoritmos, encapsular cada uma delas e torná-las intercambiáveis, **Strategy** permite que o algoritmo varie independente dos clientes que o utilizam

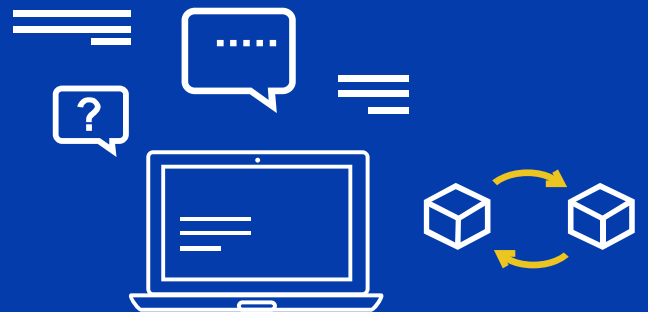


Strategy:
- Define uma interface comum para todos os algoritmos suportados

ConcreteStrategy:
- Implementa o algoritmo usando a interface do strategy

Context:
- Contém objeto da interface strategy

GATILHO

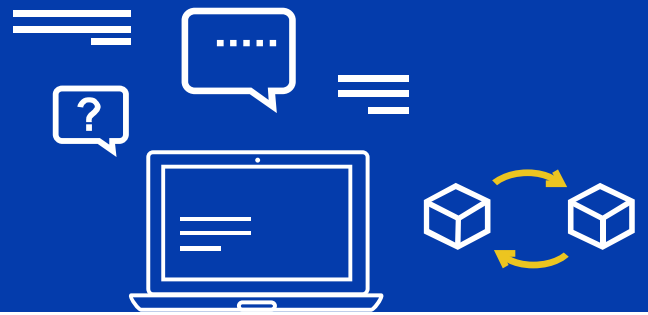


DIFERENTES VARIAÇÕES DE UM MESMO ALGORITMO

*Strategy é um **padrão comportamental de objetos** (utiliza á composição de objetos em vez de herança)*

-Se preocupa com o encapsulamento do comportamento de um objeto e com a delegação de solicitações para ele

Orientação a objetos



COMPOSIÇÃO

- Implementação de uma classe que representa um objeto quadrado
- Quadrado é composto de pontos
- Ponto é uma classe
- Logo quadrado é construída através da composição de quatro classes ponto
- Relacionamento (TEM UM)

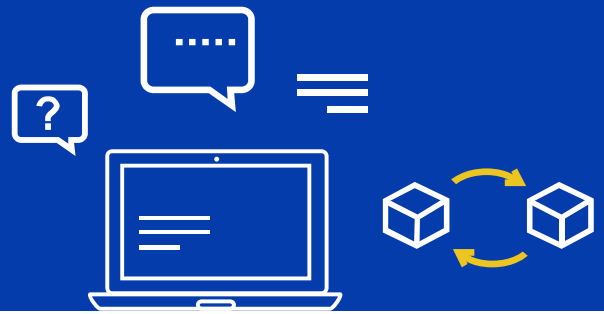
POLIMORFISMO

- Significa muitas formas , capacidade de um objeto ser referenciado de várias formas
-Vinculação dinâmica (decisão sobre qual comportamento utilizar é tomada em tempo de execução)

CLASSE ABSTRATA

- Classes Abstratas correspondem a especificações genéricas, que deverão ser concretizadas em classes derivadas

Orientação a objetos



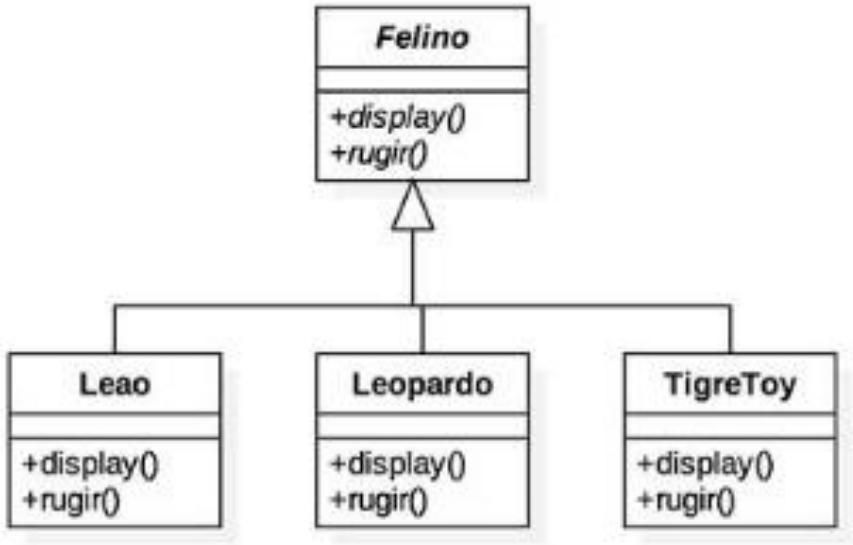
INTERFACE

- Pode ser vista como um " contrato" onde quem assina (classes que implementam) se responsabiliza por implementar os métodos da mesma

MOTIVAÇÃO

Problema: Utilizar parte de um game , concentrando-se na representação do tipo felino (existem felinos reais e felino de brinquedo)

Segue a baixo o diagrama de classes que representa essa parte do game



Adicionar outros dois comportamentos aos felinos, (**correr e pular**)

Possíveis Soluções

Mudar o tipo do comportamento em tempo de execução ????

1 - Adicionar os métodos correr e pular na classe abstrata

2- Podemos criar interfaces para que esses comportamentos existam somente nas subclasses que precisam deles

Desvantagens

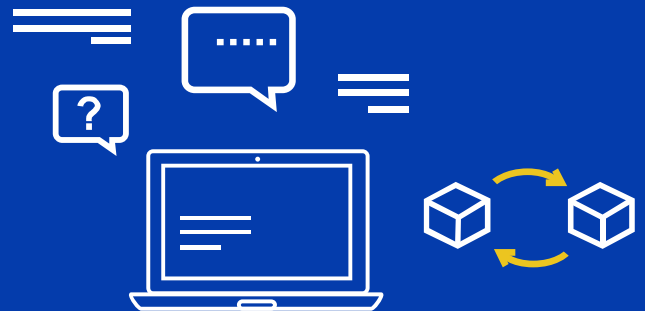
Abordagens

1 – Trabalhar com interfaces perderiamos a flexibilidade do código, pois o trecho em que os métodos correr() e pular () forem utilizados será necessário a identificação do tipo do objeto, para que não sejam invocados esses métodos em instâncias que os não tenham

2- Nessa abordagem qualquer outra nova funcionalidade implica em refatorar o projeto inteiro com novos métodos na superclasse Felino (incluindo algumas subclasses com implementações próprias) ou novas interfaces a serem implementadas

3- Ambas as opções citadas ferem o princípio de orientação a objetos "Dê prioridade a composição ao invés da herança.

Padrão Strategy



Criar famílias de algoritmos (classes) para os comportamentos **correr** e **pular**.

Vantagens

Strategy

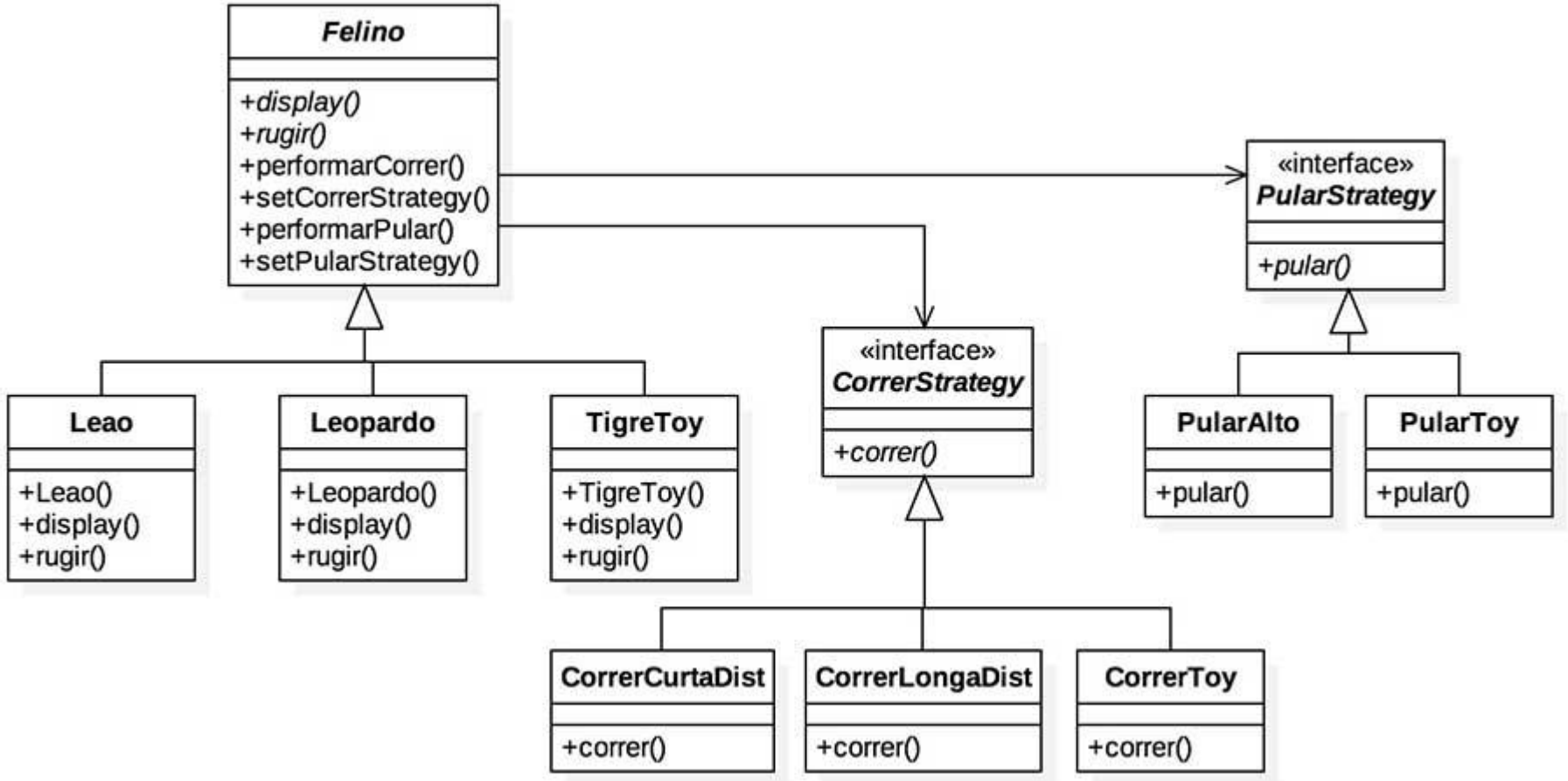
1- A reutilização de código é evidente, sendo que as famílias de algoritmos podem ser utilizadas por classes clientes de diferentes contextos, por exemplo, classe de contexto "humano" pode-se utilizar as classes correr e pular

2- Permite uma evolução de projeto mais eficiente, pois o foco é na composição, evitando que classes e subclasses clientes sejam alteradas caso, novas funcionalidades sejam adicionadas ou atualizadas

3- A possibilidade de mudança de comportamento em tempo de execução da maior dinamicidade ao projeto

Diagrama usando o padrão

Strategy



FIM