

AULA 2 – Padrão de Projetos

Prof. Igor Justino Rodrigues
17/10/2024



OBJETIVOS DA AULA

- Compreender a importância da utilização de padrões de projeto no desenvolvimento de software.
- Discutir os principais benefícios de aplicar padrões, como a reutilização de soluções, manutenção de código e otimização de processos de desenvolvimento.

17/10/2024



Cenário 1

- Imagine que você tem um sistema de vendas onde o preço de um produto específico deve ser sempre o mesmo em todas as partes do sistema, seja na página de produto, no carrinho de compras ou na área de pagamentos. O problema é que, sem um controle único desse valor, cada parte do sistema poderia acabar com uma versão diferente do preço, especialmente se alguém altera esse valor enquanto outras áreas ainda usam o antigo.

E agora?

- O Singleton resolve esse problema garantindo que o sistema tenha apenas uma única instância do valor do produto. Assim, qualquer alteração no preço é imediatamente refletida em todas as partes do sistema. Com essa abordagem, evitamos inconsistências e garantimos que o valor exibido ao cliente seja sempre o mesmo, independentemente de onde ele esteja no fluxo de compra.



VAMOS
PRATICAR E
ENTENDER!

The image shows a chalkboard with handwritten mathematical derivations. At the top left, a graph shows a curve $y = g(x)$ with a secant line and a tangent line. The secant line is labeled "Secant Lines" and the tangent line is labeled "Tangent Line". Below the graph, the expression $x+h$ is written. To the right, the derivative of f is defined as $f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$. Below this, the derivative of $f(x) = x^2$ is calculated: $f'(x) = \lim_{h \rightarrow 0} \frac{(x+h)^2 - x^2}{h} = \lim_{h \rightarrow 0} \frac{x^2 + 2xh + h^2 - x^2}{h} = \lim_{h \rightarrow 0} \frac{2xh + h^2}{h} = \lim_{h \rightarrow 0} h(2x + h)$.

$y = g(x)$

Secant Lines

Tangent Line

$x+h$

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$
$$f(x) = \lim_{h \rightarrow 0} \frac{(x+h)^2 - x^2}{h}$$
$$= \lim_{h \rightarrow 0} \frac{x^2 + 2xh + h^2 - x^2}{h}$$
$$= \lim_{h \rightarrow 0} \frac{2xh + h^2}{h}$$
$$= \lim_{h \rightarrow 0} h(2x + h)$$

$g(x+h) - g(x)$

Vou acabar precisando relembrar...



Classe: Estrutura que define as características e comportamentos (propriedades e métodos) de um objeto. Serve como um "molde" para criar instâncias.



Objeto: Representação de uma instância de uma classe, contendo valores específicos para as propriedades e podendo executar os métodos da classe.



Instância: Um objeto criado a partir de uma classe. Cada instância pode ter seus próprios valores para as propriedades definidas pela classe.



Atributo/Propriedade: Dados armazenados em uma classe ou objeto que representam suas características, como a "cor" de um carro ou o "valor" de um produto.



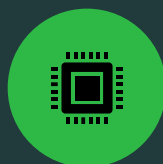
Método: Função ou procedimento definido dentro de uma classe que descreve uma ação que uma instância da classe pode realizar.



Variável: Espaço na memória que armazena dados, como números ou textos, identificados por um nome. Pode conter valores modificáveis.



Parâmetro: Variável usada em um método ou função para receber valores de entrada, permitindo que o método funcione com dados específicos quando chamado.



API (Application Programming Interface): Conjunto de funções, classes, ou métodos que permitem que diferentes partes de um programa, ou programas distintos, comuniquem-se entre si.

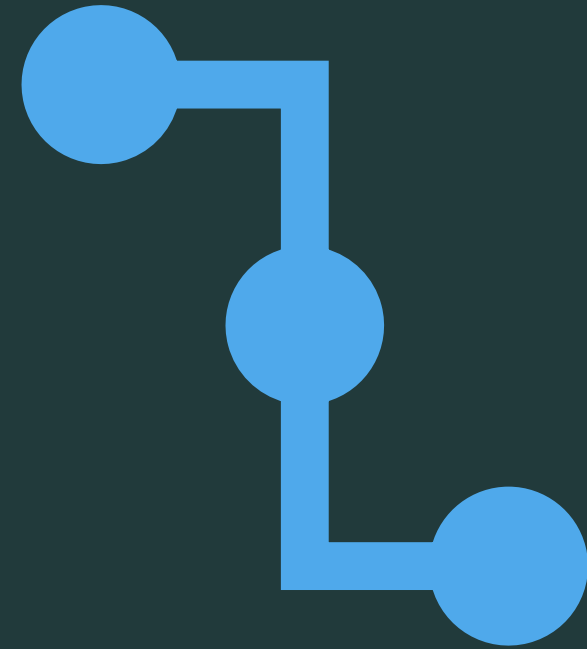
Cenário 2

- Imagine uma loja online que permite aos gerentes ajustar o valor dos produtos com base em políticas de desconto ou aumento. Em épocas de promoção, os gerentes aplicam um desconto percentual para atrair mais clientes, enquanto, em períodos de alta demanda, aplicam um aumento de preço para maximizar o lucro. No entanto, esses ajustes devem ser aplicados de forma flexível, dependendo da situação, sem a necessidade de alterar o código principal da aplicação a cada nova política de preço.



Como vou fazer isso...

O problema aqui é garantir que o sistema seja capaz de aplicar diferentes estratégias de modificação de valor (como desconto e aumento) de forma intercambiável e flexível, sem comprometer a estrutura da classe principal do produto. Assim, o uso do padrão **Strategy** é uma solução eficiente para modificar o valor de um produto dinamicamente, apenas alterando a estratégia aplicada, mantendo a classe Produto flexível e fácil de adaptar para futuras necessidades.



VAMOS
PRATICAR E
ENTENDER!



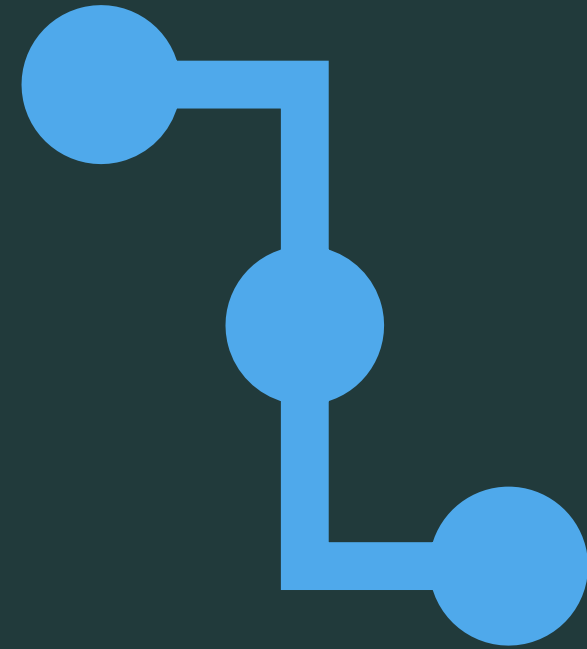


Cenário 3

- Imagine uma loja online que oferece diferentes tipos de produtos, como eletrônicos, roupas e alimentos. Cada categoria de produto possui características específicas e requer tratamentos diferentes, como cálculos de frete, políticas de desconto e condições de armazenamento.
- Para gerenciar a criação desses produtos de forma organizada e evitar que o código principal precise conhecer os detalhes de cada tipo específico, a loja precisa de uma maneira flexível de criar os objetos certos para cada categoria de produto. Isso permitirá que novas categorias sejam adicionadas no futuro sem a necessidade de modificar o código central da aplicação.
- O problema é encontrar uma forma de criar esses produtos específicos de maneira simplificada e centralizada, para que o sistema saiba como produzir o objeto correto para cada tipo de produto, conforme a necessidade.

Aí lascou foi tudo...

Uma necessidade importante aqui é garantir que o sistema seja capaz de criar diferentes tipos de produtos (como eletrônicos, roupas e alimentos) de forma flexível e organizada, sem a necessidade de modificar o código principal toda vez que um novo tipo for adicionado. O uso do padrão **Factory** oferece uma solução eficiente para centralizar a criação desses produtos, permitindo que o sistema decida automaticamente qual tipo específico de produto deve ser instanciado com base na necessidade, mantendo a aplicação modular e fácil de expandir para novas categorias.



```
mirror_mod = modifier_ob.  
Set mirror object to mirror  
mirror_mod.mirror_object  
operation == "MIRROR X":  
mirror_mod.use_x =  
mirror_mod.use_y =  
mirror_mod.use_z =  
operation == "MIRROR Y":  
mirror_mod.use_x = False  
mirror_mod.use_y = True  
mirror_mod.use_z = False  
operation == "MIRROR Z":  
mirror_mod.use_x = False  
mirror_mod.use_y = False  
mirror_mod.use_z = True
```

```
selection at the end -add  
mirror_ob.select= 1  
modifier_ob.select=1  
context.scene.objects.active  
("Selected" + str(modifier  
mirror_ob.select = 0  
= bpy.context.selected_obj  
data.objects[one.name].select  
print("please select exactly
```


```
-- OPERATOR CLASSES --
```

```
types.Operator):  
X mirror to the selected  
object.mirror_mirror_x"  
mirror X"
```


```
context):  
context.active_object is not
```



VAMOS
PRATICAR E
ENTENDER!



Resumo dos padrões estudados



Singleton: Centraliza o valor do produto em uma única instância, permitindo leitura e atualização.



Strategy: Aplica diferentes estratégias de alteração no valor do produto, como desconto ou aumento.



Factory: Cria dinamicamente a estratégia desejada para aplicar a alteração no valor.