



Design Pattern

Mateus Barbosa Martins

Builder

O Design Pattern Builder é uma solução elegante para construir objetos complexos com uma interface simples. Ele separa a construção de um objeto de sua representação, permitindo que você crie diferentes variações de um objeto a partir de uma única interface.



Simplicidade



Flexibilidade



Reutilização

Builder

Imagine que você está construindo um sistema de e-commerce e precisa criar um objeto Produto com atributos complexos, como nome, preço, descrição, imagem, etc. O Builder pode ser usado para facilitar este processo:

- O Builder encapsula a lógica de construção do objeto, tornando o código mais organizado.
- O uso de métodos de configuração permite uma construção passo a passo, adaptando-se às necessidades.
- O Builder pode ser usado para criar múltiplos objetos com configurações diferentes, sem a necessidade de criar classes separadas para cada variação.

```
1 class Produto:
2     def __init__(self, nome, preco, descricao, imagem):
3         self.nome = nome
4         self.preco = preco
5         self.descricao = descricao
6         self.imagem = imagem
7
8 class ProdutoBuilder:
9     def __init__(self):
10         self.produto = Produto("", 0, "", "")
11
12     def set_nome(self, nome):
13         self.produto.nome = nome
14         return self
15
16     def set_preco(self, preco):
17         self.produto.preco = preco
18         return self
19
20     def set_descricao(self, descricao):
21         self.produto.descricao = descricao
22         return self
23
24     def set_imagem(self, imagem):
25         self.produto.imagem = imagem
26         return self
27
28     def build(self):
29         return self.produto
30
31 # Usando o builder para criar um produto
32 produto = ProdutoBuilder() \
33     .set_nome("Camiseta") \
34     .set_preco(29.99) \
35     .set_descricao("Camiseta de algodão, tamanho M") \
36     .set_imagem("camiseta.jpg") \
37     .build()
38
39 print(produto.nome) # Saída: Camiseta
40 print(produto.preco) # Saída: 29.99
```

Prototype

O Design Pattern Prototype é uma solução para criar novos objetos a partir de um modelo já existente (o protótipo), sem precisar reescrever todo o processo de criação. Ele é ideal para situações onde a criação de novos objetos é complexa ou requer muitas etapas.



Reutilização



Eficiência



Simplicidade

Prototype

Imagine que você está criando um sistema de gerenciamento de usuários e precisa criar novos usuários com base em um perfil padrão. O Prototype pode ser usado para clonar o perfil padrão e, em seguida, ajustar as propriedades do novo usuário:

- O Prototype encapsula a lógica de criação de novos objetos, simplificando o processo.
- O método `clone()` fornece uma forma eficiente de criar cópias do objeto, sem a necessidade de reescrever todo o código.
- O Prototype permite criar diferentes variações de um objeto a partir de um modelo base, ajustando apenas as propriedades necessárias.

```
1 class Usuario:
2     def __init__(self, nome, email, senha):
3         self.nome = nome
4         self.email = email
5         self.senha = senha
6
7     def clone(self):
8         return Usuario(self.nome, self.email, self.senha)
9
10 # Criando o perfil padrão
11 perfil_padrao = Usuario("Padrão", "padrao@example.com", "123456")
12
13 # Criando um novo usuário a partir do perfil padrão
14 novo_usuario = perfil_padrao.clone()
15 novo_usuario.nome = "João da Silva"
16 novo_usuario.email = "joao.silva@example.com"
17
18 print(novo_usuario.nome) # Saída: João da Silva
19 print(novo_usuario.email) # Saída: joao.silva@example.com
```