Questão 4

Padrões de Projeto em Engenharia de Software

O que são Padrões de Projeto?

Os padrões de projeto são soluções reutilizáveis para problemas comuns que ocorrem no desenvolvimento de software. Eles fornecem uma forma estruturada de resolver problemas de design, oferecendo uma abordagem testada e comprovada que pode ser aplicada a situações semelhantes. Esses padrões ajudam a melhorar a qualidade do software, facilitando a comunicação entre desenvolvedores e promovendo a reutilização de código.

Relação com a Atividade de Evolução de Software

A evolução de software é o processo contínuo de adaptação e melhoria de um sistema de software após sua entrega inicial. Padrões de projeto desempenham um papel crucial na evolução de software por vários motivos:

1. Manutenção: Padrões de projeto bem aplicados tornam o código mais compreensível e estruturado, facilitando a manutenção e a modificação do software ao longo do tempo.
2. Extensibilidade: Eles promovem a criação de sistemas flexíveis que podem ser facilmente estendidos com novas funcionalidades sem a necessidade de grandes reescritas de código.
3. Reutilização: Padrões de projeto encorajam a reutilização de soluções bem definidas, reduzindo o esforço necessário para implementar funcionalidades semelhantes em diferentes partes do software ou em diferentes projetos.
4. Comunicação: O uso de padrões de projeto facilita a comunicação entre os membros da equipe de desenvolvimento, uma vez que os padrões fornecem um vocabulário comum e uma compreensão compartilhada das soluções de design.
5. Refatoração: Durante o processo de evolução, o código frequentemente precisa ser refatorado. Padrões de projeto fornecem diretrizes claras para refatoração, ajudando a manter a integridade e a coesão do design do software.

Padrão de Projeto Strategy

Descrição

O padrão de projeto Strategy define uma família de algoritmos, encapsula cada um deles e os torna intercambiáveis. Este padrão permite que o algoritmo varie independentemente

dos clientes que o utilizam. Em outras palavras, o padrão Strategy permite que uma classe se comporte de maneira diferente com base em qual algoritmo de comportamento é escolhido.

Este padrão é especialmente útil quando uma classe tem muitos comportamentos relacionados que podem ser variáveis de acordo com diferentes condições ou situações. Ao encapsular esses comportamentos em classes separadas (estratégias), o código se torna mais flexível e mais fácil de manter.

Estrutura

A estrutura do padrão Strategy envolve os seguintes componentes principais:

1. Contexto: A classe que utiliza uma estratégia e é configurada com um objeto Strategy concreto.
2. Strategy: A interface comum para todos os algoritmos suportados pelo Contexto. Declara um método que todos os algoritmos devem implementar.
3. ConcreteStrategy: Classes que implementam a interface Strategy, fornecendo implementações específicas dos algoritmos.

A interação entre esses componentes permite que o Contexto altere seu comportamento dinamicamente ao mudar a ConcreteStrategy que está sendo utilizada.

Exemplo de Implementação do Padrão Strategy

Considere um exemplo de cálculo de preços com diferentes estratégias de desconto. Em vez de ter múltiplas condições dentro da classe que calcula o preço, podemos definir diferentes estratégias de desconto e aplicá-las conforme necessário.

```python
# Interface Strategy
class DiscountStrategy:
    def calculate(self, price):
        pass

//ConcreteStrategy A: Sem desconto
class NoDiscount(DiscountStrategy):
    def calculate(self, price):
        return price

//ConcreteStrategy B: Desconto de 10%
class TenPercentDiscount(DiscountStrategy):
```

```
    def calculate(self, price):
        return price * 0.9

//ConcreteStrategy C: Desconto de 20%
class TwentyPercentDiscount(DiscountStrategy):
    def calculate(self, price):
        return price * 0.8

//Contexto
class PriceCalculator:
    def __init__(self, strategy: DiscountStrategy):
        self.strategy = strategy

    def calculate_price(self, price):
        return self.strategy.calculate(price)

//Uso do padrão Strategy
price = 100
calculator = PriceCalculator(NoDiscount())
print(f"Preço sem desconto: {calculator.calculate_price(price)}")

calculator.strategy = TenPercentDiscount()
print(f"Preço com 10% de desconto: {calculator.calculate_price(price)}")

calculator.strategy = TwentyPercentDiscount()
print(f"Preço com 20% de desconto: {calculator.calculate_price(price)}")
```

Neste exemplo, a classe `PriceCalculator` utiliza uma estratégia de desconto configurada no momento da criação ou alterada posteriormente, permitindo uma grande flexibilidade na aplicação de diferentes estratégias de desconto.