INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA - CÂMPUS GASPAR
ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

CAIO BERNARDO PEREIRA
CAIO BERNARDO PEREIRA

APLICAÇÃO DE DESIGN PATTERNS EM UM SISTEMA DE GERENCIAMENTO DE JOGOS DE FUTEBOL

PADRÕES DE PROJETO DE SOFTWARE

GASPAR

2024

SUMÁRIO

1 INTRODUÇÃO

Neste trabalho, exploramos a aplicação de padrões de projeto (design patterns) em um sistema de gerenciamento de competições esportivas, utilizando o exemplo de uma final de futebol. Os padrões de projeto escolhidos: Singleton; Builder; Composite; Observer. São amplamente reconhecidos na engenharia de software por sua capacidade de resolver problemas recorrentes de design e promover um código mais modular, reutilizável e manutenível. Ao implementar esses padrões, o objetivo é demonstrar como eles podem ser utilizados para estruturar o código de forma eficiente, garantindo que o sistema possa evoluir de maneira sustentável à medida que novas funcionalidades são adicionadas.

2 SINGLETON

O padrão Singleton é utilizado para garantir que apenas uma instância da classe GerenciadorDeCompeticoes seja criada, garantindo que todas as partes do sistema compartilhem o mesmo estado do campeonato. Este padrão é particularmente útil em sistemas onde uma única configuração global ou um ponto de acesso a recursos é necessário.

No contexto do gerenciamento de competições, é fundamental que o controle do campeonato seja centralizado, evitando inconsistências e garantindo que todas as operações ocorram em uma única instância controlada.

• Exemplo de Implementação:

```
class SingletonMeta(type):
    _instances = {}

    def __call__(cls, *args, **kwargs):
        if cls not in cls._instances:
            instance = super().__call__(*args, **kwargs)
            cls._instances[cls] = instance
        return cls._instances[cls]

class GerenciadorDeCompetic o es(metaclass=SingletonMeta):
    pass
```

Listing 2.1 – Singleton utilizado

O padrão Singleton foi utilizado para garantir que a classe GerenciadorDeCompetições tenha apenas uma única instância durante toda a execução do programa, centralizando o gerenciamento do estado global da aplicação.

3 BUILDER

O padrão Builder é utilizado para a criação de objetos complexos, como os times de futebol, que incluem vários jogadores e um técnico. Este padrão facilita a construção de objetos complexos ao dividir o processo de construção em etapas menores e mais gerenciáveis.

Ao construir um time de futebol, existem vários componentes que precisam ser configurados, como a seleção de jogadores, técnico e reservas. O Builder permite que essa construção seja flexível e extensível, possibilitando a criação de diferentes configurações de times sem modificar o código base.

• Exemplo de Implementação:

```python
# builder.py
class ConstrutorDeTime:
    def __init__(self):
        self.time = Time()

    def adicionar_jogador(self, nome, posicao):
        self.time.jogadores.append(Jogador(nome, posicao))
        return self

    def adicionar_tecnico(self, nome):
        self.time.tecnico = Tecnico(nome)
        return self

    def build(self):
        return self.time

# Exemplo de uso
construtor = ConstrutorDeTime()
flamengo = (construtor.adicionar_tecnico("Jorge Jesus")
            .adicionar_jogador("Diego Alves", "Goleiro")
            .adicionar_jogador("Rafinha", "Lateral Direito")
            .build())
```

Listing 3.1 – Builder

O padrão Builder foi utilizado para facilitar a construção de objetos complexos como o Time, permitindo que diferentes configurações de times sejam criadas de forma flexível e passo a passo.

4 COMPOSITE

O padrão Composite é utilizado para representar estruturas hierárquicas de objetos, como a composição de um time de futebol que inclui jogadores titulares e reservas. Esse padrão permite tratar objetos individuais e composições de objetos de forma uniforme.

Em um time de futebol, os jogadores titulares e reservas podem ser tratados como componentes individuais, mas também como parte de um conjunto maior (o time). O Composite facilita a manipulação dessas estruturas complexas de forma coesa e simplificada.

```
# composite.py
class Time:
    def __init__(self):
        self.jogadores = []
        self.tecnico = None

    def adicionar(self, membro):
        self.jogadores.append(membro)

    def exibir_componentes(self):
        print(f"T cnico: {self.tecnico.nome}")
        for jogador in self.jogadores:
            print(f"Jogador: {jogador.nome} - Posi   o: {
   jogador.posicao}")
```

Listing 4.1 – Composite

O padrão Builder foi utilizado para facilitar a construção de objetos complexos como o Time, permitindo que diferentes configurações de times sejam criadas de forma flexível e passo a passo.

5 OBSERVER

O padrão Observer é utilizado para notificar os componentes do sistema sobre eventos específicos, como a ocorrência de um gol durante uma partida. Ele é ideal para cenários onde múltiplos objetos precisam ser informados sobre mudanças em outro objeto.

Durante uma partida de futebol, é importante que diferentes partes do sistema (como a exibição de placar e os logs de eventos) sejam atualizadas imediatamente quando um gol é marcado. O Observer permite que essa atualização ocorra de maneira automática e desacoplada, melhorando a coesão e facilitando a manutenção do código.

```
1  class Jogo:
2      def __init__(self):
3          self._observers = []
4
5      def adicionar_observer(self, observer):
6          self._observers.append(observer)
7
8      def notificar_gol(self, time, jogador, minuto):
9          for observer in self._observers:
10             observer.atualizar(time, jogador, minuto)
```

Listing 5.1 – Observer

O padrão Observer foi implementado para que diferentes partes do sistema sejam notificadas de eventos de gol, permitindo uma atualização automática e desacoplada das informações de jogo.

6 CONCLUSÃO

A implementação dos padrões de projeto Singleton, Builder, Composite, e Observer no contexto do gerenciamento de competições esportivas mostrou-se eficaz na criação de uma arquitetura de software robusta e flexível. Cada padrão desempenhou um papel crucial na resolução de desafios específicos, como a construção de times de futebol, a gestão de partidas e a comunicação de eventos em tempo real, como gols. O uso desses padrões não só facilitou o desenvolvimento do sistema, mas também assegurou que o código fosse facilmente adaptável a futuras mudanças e expansões. Isso reforça a importância dos design patterns na criação de soluções de software que são tanto eficientes quanto fáceis de manter.

7 REFERENCIAS

• Refactoring Guru. Padrões de Projeto. Disponível em: https://refactoring.guru/pt-br/design- patterns. Acesso em: 10/08/2024.

• Refactoring Guru. Catálogo de Padrões de Projeto. Disponível em: https://refactoring.guru/pt-br/design-patterns/catalog. Acesso em: 10/08/2024.

• Refactoring Guru. Padrão de Projeto Builder. Disponível em: https://refactoring.guru/pt-br/design-patterns/builder. Acesso em: 11/08/2024.

• Refactoring Guru. Exemplo de Padrão Builder em Python. Disponível em: https://refactoring.guru/pt-br/design-patterns/builder/python/example. Acesso em: 11/08/2024.

• Refactoring Guru. Padrão de Projeto Singleton. Disponível em: https://refactoring.guru/pt-br/design-patterns/singleton. Acesso em: 12/08/2024.

• Refactoring Guru. Exemplo de Padrão Singleton em Python. Disponível em: https://refactoring.guru/pt-br/design-patterns/singleton/python/example. Acesso em: 12/08/2024.

• Refactoring Guru. Padrão de Projeto Composite. Disponível em: https://refactoring.guru/pt-br/design-patterns/composite. Acesso em: 13/08/2024.

• Refactoring Guru. Exemplo de Padrão Composite em Python. Disponível em: https://refactoring.guru/pt-br/design-patterns/composite/python/example. Acesso em: 13/08/2024.

• Refactoring Guru. Padrão de Projeto Observer. Disponível em: https://refactoring.guru/pt-br/design-patterns/observer. Acesso em: 14/08/2024.

• Refactoring Guru. Exemplo de Padrão Observer em Python. Disponível em: https://refactoring.guru/pt-br/design-patterns/observer/python/example. Acesso em: 14/08/2024.

• Croc, R. Programação Paralela e Distribuída 2007/08. Disponível em: https://www.dcc.fc.up.pt/ri- croc/aulas/0708/ppd/apontamentos/fundamentos.pdf. Acesso em: 15/08/2024.