N_JOG DIG2 - Texto de apoio

Site: EAD Mackenzie
Tema: JOGOS DIGITAIS {TURMA 04B} 2023/2
Livro: N_JOG DIG2 - Texto de apoio

Impresso por: FELIPE BALDIM GUERRA .
Data: quinta, 31 ago 2023, 01:51

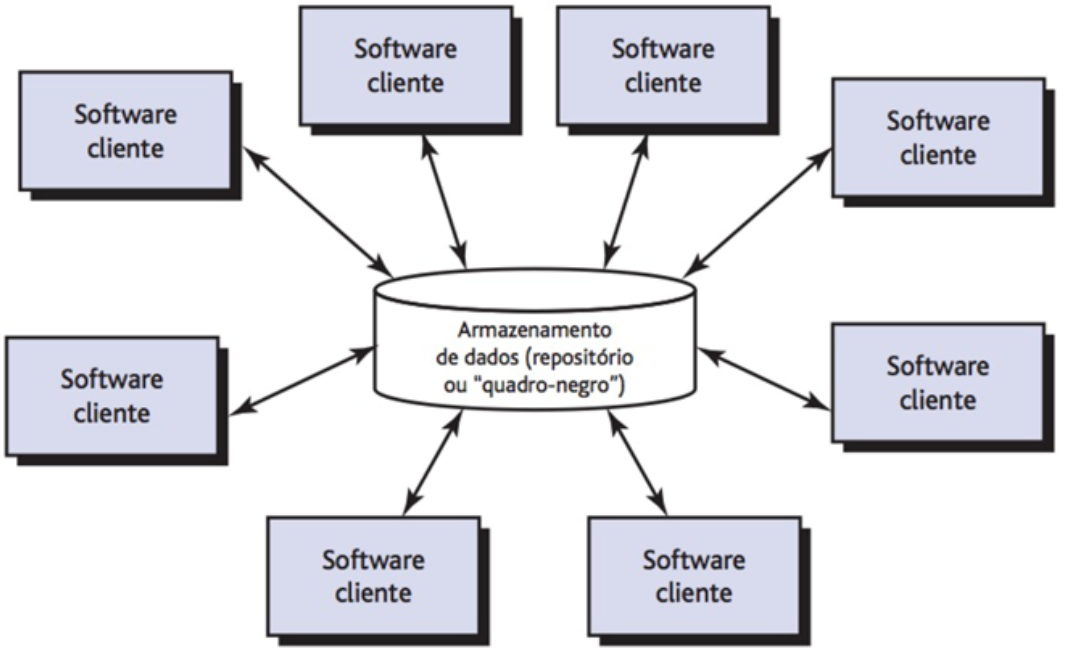
Descrição

Índice

1. O QUE É ARQUITETURA DE SOFTWARE?

Um projeto de arquitetura está preocupado em como organizar um sistema e em como deve ser sua estrutura. Em sistemas tradicionais (ERP, aplicações web, entre outros, ou seja, sistemas não jogos – em sua maioria), a arquitetura de software está preocupada em criar um modelo abstrato do sistema em questão, apresentando os diversos componentes que o formam e as maneiras como tais componentes interagem entre si.

Figura 1 – Modelo de Arquitetura de um sistema centralizado em dados Bruce



Fonte: PRESSMAN (2016).

Podemos projetar arquiteturas de software em dois níveis: pequena ou grande escala.

- **Pequena escala:** uma arquitetura preocupada em programas individuais. Nesse nível, estamos interessados em como um software, individualmente, deve ser organizado, em como seus componentes internos devem interagir entre si. Exemplos desse tipo de arquitetura são os modelos de aplicações Web (que seguem padrões MVC, por exemplo).

- **Grande escala:** nesse tipo de arquitetura, estamos interessados nos chamados sistemas corporativos complexos e de grande escala, nos quais cada sistema interage direta ou indiretamente com vários outros sistemas. São relevantes, nesse nível, as definições de como esses sistemas se comunicam e as regras de negócio geral presentes nesse ambiente. Por exemplo, um sistema de vendas online precisa integrar-se ao sistema de controle de estoque, sistema de controle de logística etc.

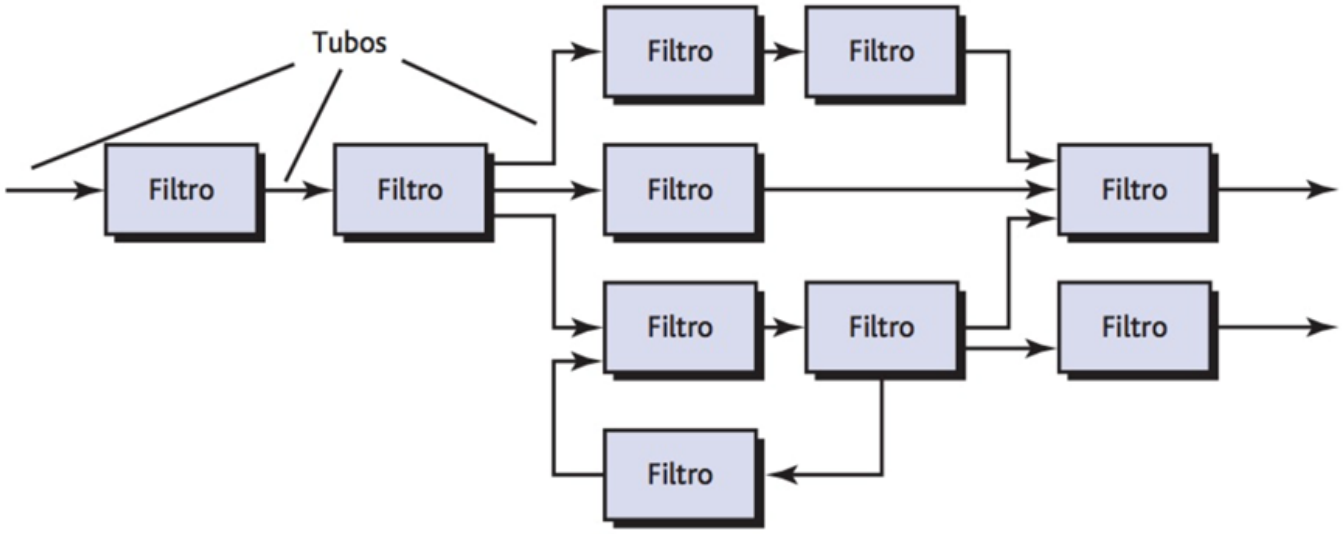
Softwares criados para ambientes computacionais também apresentam estilos de arquitetura. Esses estilos descrevem ou seguem categorias globais de sistemas. Por exemplo, sistemas semelhantes, independentemente da empresa ou do ambiente em que estejam instalados, executam funções semelhantes, como um banco de dados que realiza uma função de armazenamento de dados exigida por um sistema.

Normalmente, seguimos padrões específicos de arquiteturas para facilitar o trabalho de análise, implementação e evolução do sistema. Esses padrões podem ser utilizados como forma de dar uma estrutura global ao sistema.

Neste aspecto, temos vários padrões arquiteturais pré-definidos, cada um para um propósito específico. Por exemplo:

- Arquiteturas de fluxo de dados: São aplicadas quando dados de entrada devem ser transformados por meio de uma série de componentes computacionais ou de manipulação em dados de saída. Nesse caso, utilizamos uma série de componentes chamados filtros que podem ser empregados de forma serial, paralela etc.

Figura 2 – Arquitetura de fluxo de dados

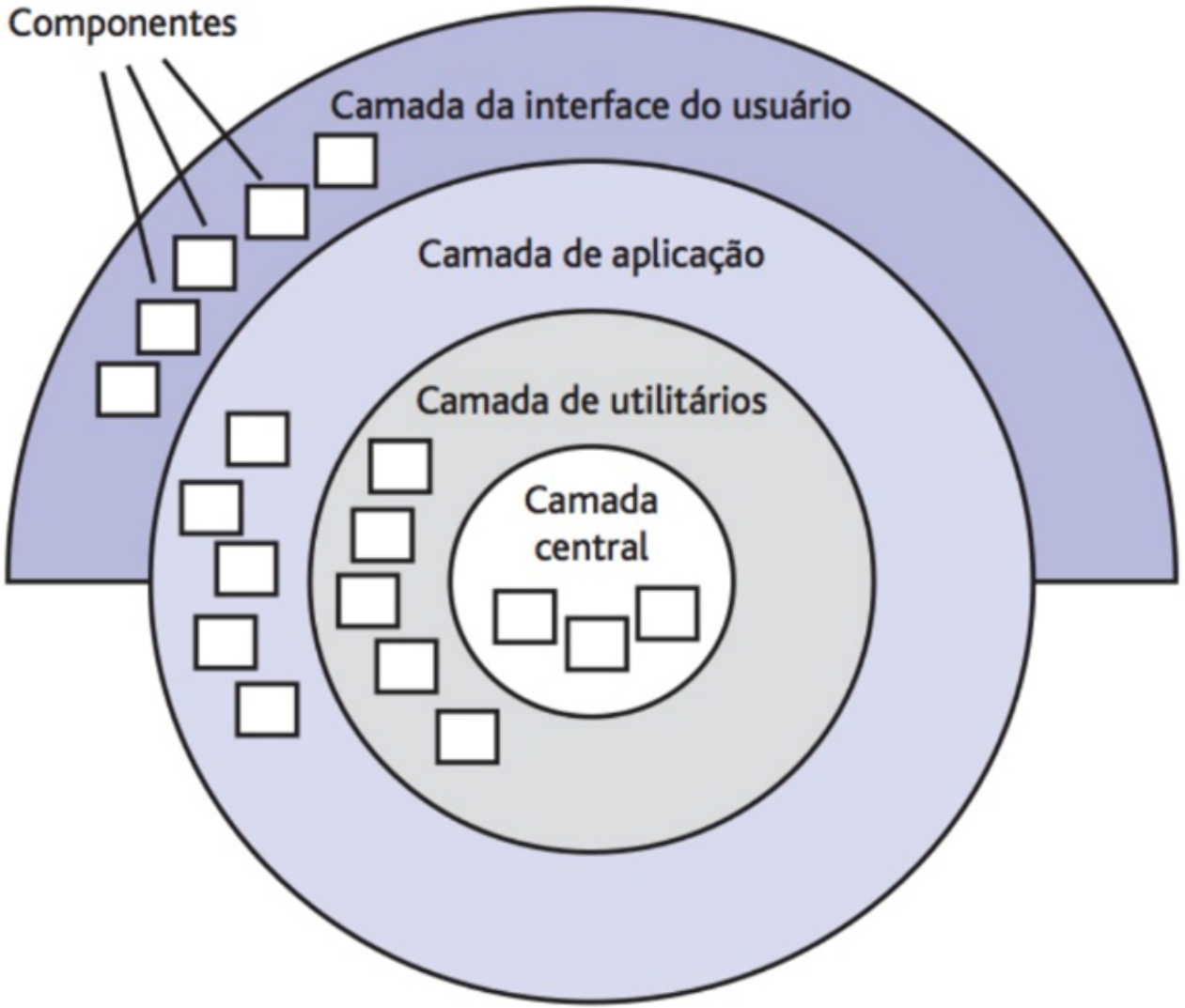


Tubos e filtros

Fonte: PRESSMAN (2016).

- Arquiteturas orientadas a objetos: Os componentes de um sistema neste estilo encapsulam os dados e as operações que devem ser aplicadas para manipular esses dados.
- Arquiteturas em camadas: Nesta arquitetura, são definidas várias camadas diferentes, cada uma realizando operações que progressivamente se tornam mais próximas do conjunto de instruções de máquina. Na camada mais externa, os componentes atendem às operações da interface do usuário. Na camada mais interna, fazem a interface com o sistema operacional. As camadas intermediárias fornecem serviços utilitários e funções de software de aplicação.

Figura 3 – Arquitetura em camadas



Fonte: PRESSMAN (2016).

2. PADRÕES DE PROJETO

Como nosso foco nesta disciplina é o desenvolvimento de jogos, estudaremos mais a fundo as arquiteturas de pequena escala.

Assim como visto anteriormente nas arquiteturas de larga escala, temos uma série de padrões arquiteturais, chamados especificamente de padrões de projeto.

Esses padrões de projeto, normalmente, são associados a um projeto orientado a objetos e, desta forma, a linguagens de programação que também suportam a orientação a objetos. Tais padrões definem estratégias de implementações utilizando mecanismos tradicionais desse paradigma como herança, polimorfismo etc.

Os padrões de projeto são organizados sob um nome, como forma de categorização e descrição do problema que estão propondo resolver, a fim de ser apresentada uma possível solução.

Afinal, todos nós já nos deparamos com um problema de projeto e, silenciosamente, pensamos: “será que alguém já desenvolveu uma solução para isso?” A resposta é quase sempre “sim”! O desafio é encontrar a solução; garantir que, de fato, ela se adapte ao problema em questão; entender as restrições que talvez limitem a maneira pela qual a solução é aplicada e, por fim, traduzir a solução proposta para seu ambiente de projeto.

Quadro 1 – Panorama

PANORAMA

O que é?

O projeto baseado em padrões cria uma nova aplicação por meio da busca de um conjunto de soluções comprovadas para um conjunto de problemas claramente delineados. Cada problema e sua respectiva solução são descritos por um padrão de projeto que foi catalogado e investigado por outros engenheiros de software, os quais se depararam com o problema e implementaram a solução ao projetarem outras aplicações. Cada padrão de projeto nos oferece uma abordagem comprovada para parte do problema a ser resolvido.

Quem realiza?

Um engenheiro de software examina cada problema que surge para uma nova aplicação e tenta encontrar uma solução relevante por meio de pesquisa em um ou mais repositórios de padrões.

Por que é importante?

Você já ouviu a expressão “reinventar a roda”? Ela ocorre toda hora em desenvolvimento de software e é uma perda de tempo e energia. Ao usarmos padrões de projeto, podemos encontrar uma solução comprovada para um problema específico. À medida que cada padrão é aplicado, são integradas soluções, e a aplicação a ser construída se aproxima cada vez mais de um projeto completo.

Quais são as etapas envolvidas?

O modelo de requisitos é examinado para isolar o conjunto hierárquico de problemas a serem resolvidos. O espaço de problemas é subdividido de modo que subconjuntos de problemas associados a funções e características específicas de software possam ser identificados. Os problemas também podem ser organizados por tipo: de arquitetura, de componentes, algorítmicos, de interfaces do usuário etc. Uma vez definido um subconjunto de problemas, pesquisam-se um ou mais repositórios de padrões para determinar se existe um padrão de projeto representado em um nível de abstração apropriado. Padrões aplicáveis são adaptados às necessidades específicas do software a ser construído. A solução de problemas personalizados é aplicada em situações nas quais não foi encontrado nenhum padrão.

Qual é o artefato?

É desenvolvido um modelo de projeto que representa a estrutura da arquitetura, a interface do usuário e os detalhes em nível de componentes.

Como garantir que o trabalho foi realizado corretamente?

À medida que cada padrão de projeto é traduzido em algum elemento do modelo de projeto, os artefatos são revistos em termos de clareza, correção, completude e consistência em relação aos requisitos e entre si.

Adaptado de: PRESSMAN; MAXIM (2016).

3. TIPOS DE PADRÕES

Padrões de projeto abrangem um amplo espectro de abstração e aplicação.

- Os padrões de arquitetura descrevem problemas de projeto de caráter amplo e diverso, os quais são resolvidos usando-se uma abordagem estrutural.
- Os padrões de dados descrevem tanto os problemas orientados a dados recorrentes quanto as soluções de modelagem de dados que podem ser usadas para resolvê-los.
- Os padrões de componentes (padrões de projeto) tratam dos problemas associados ao desenvolvimento de subsistemas e componentes, da maneira pela qual eles se comunicam entre si e de seu posicionamento em uma arquitetura maior.
- Os padrões de projeto para interfaces descrevem problemas comuns de interfaces do usuário e suas respectivas soluções, com um sistema de forças que inclui as características específicas dos usuários.
- Os padrões para WebApp tratam de um conjunto de problemas encontrados ao se construir WebApps e, em geral, incorporam muitas das demais categorias de padrões que acabamos de mencionar.
- Os padrões móveis descrevem os problemas comumente encontrados ao se desenvolver soluções para plataformas móveis.
- Em um nível de abstração mais baixo, os idiomas descrevem de que maneira implementar total ou parcialmente um algoritmo específico, assim como indicam a estrutura de dados para um componente de software no contexto de uma linguagem de programação específica.

3.1. Tipos de padrões – GoF

- Os padrões criacionais se concentram na “criação, composição e representação” de objetos e dispõem de mecanismos que tanto facilitam a instanciação de objetos em um sistema quanto impõem “restrições sobre o tipo e número de objetos que podem ser criados em um sistema”.

- Os padrões estruturais focalizam problemas e soluções associadas a como classes e objetos são organizados e integrados para construir uma estrutura maior.

- Os padrões comportamentais tratam de problemas associados à atribuição de responsabilidade entre objetos e à maneira como a comunicação é realizada entre objetos.

4. FRAMEWORKS

- “Miniarquitetura” reutilizável que serve como base a partir da qual outros padrões de projeto podem ser aplicados.

- Framework não é um padrão de arquitetura, mas sim um esqueleto com um conjunto de “pontos de conexão” (também chamados ganchos e encaixes) os quais permitem que este esqueleto seja adaptado a um domínio de problemas específico. Um framework inclui aspectos estáticos/ estruturais e dinâmicos/ comportamentais que dão suporte ao uso dos padrões de projeto. Os pontos de conexão possibilitam que integremos ao esqueleto classes ou funcionalidades específicas de um problema. Em um contexto orientado a objetos, um framework é um conjunto de classes que cooperam entre si.

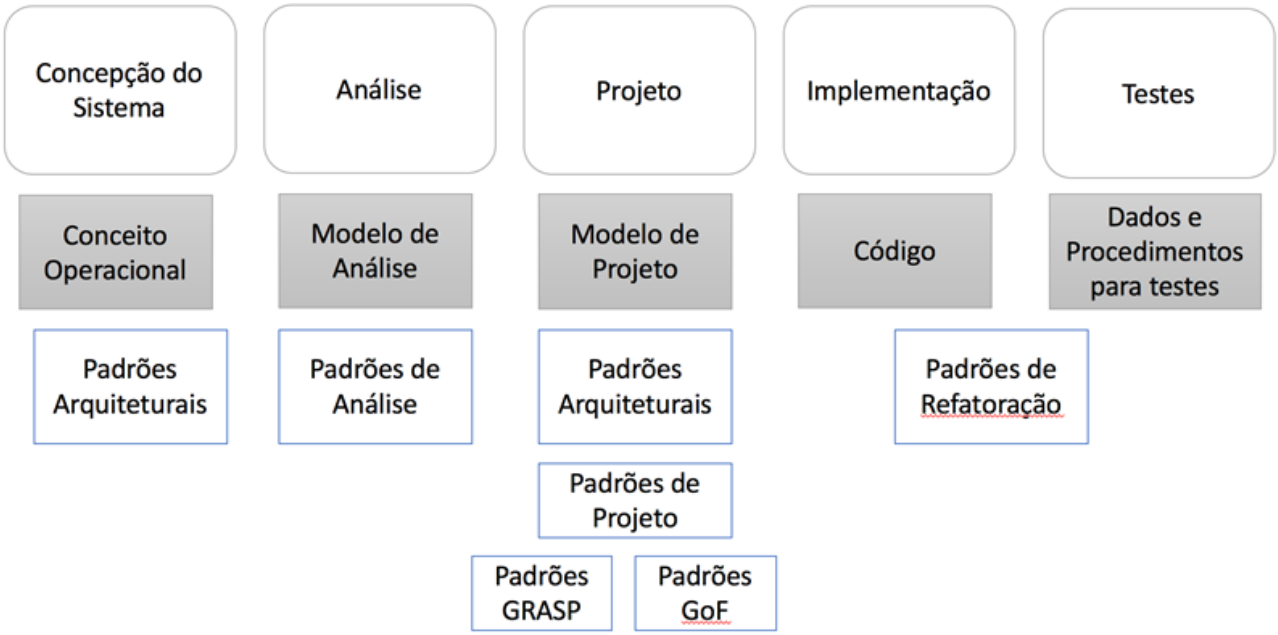
5. Frameworks versus Padrões de Projeto – GoF

- Os padrões de projeto são mais abstratos do que os frameworks. Os frameworks podem ser incorporados ao código, mas apenas exemplos de padrões podem ser incorporados ao código. Um ponto forte dos frameworks de uso de padrões é que eles podem ser escritos em linguagens de programação, e não só estudados, mas executados e reutilizados diretamente.

- Os padrões de projeto são elementos arquiteturais menores do que os frameworks. Esses frameworks podem conter vários padrões de projeto, mas a recíproca jamais é verdadeira.

- Os padrões de projeto são menos especializados do que os frameworks. Os frameworks apresentam um domínio de aplicação particular. Os padrões de projeto podem ser usados em praticamente qualquer tipo de aplicação. Embora sejam possíveis padrões de projeto mais especializados, até mesmo esses não ditariam uma arquitetura de aplicação.

Figura 4 – Sintetizando arquitetura de software e padrões de projeto



Fonte: Elaborada pelo autor.

Figura 5 – Mapa dos Padrões GOF

		Propósito		
		Criação	Estrutura	Comportamento
Escopo	Classe	Factory Method	Class Adapter	Interpreter Template Method
	Objeto	Builder Abstract Factory Prototype Singlenton	Object Adapter Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Fonte: Elaborada pelo autor.

- **Factory Method:** define uma interface para criar um objeto, mas deixa que subclasses decidam que classe instanciar.
- **Prototype:** especifica tipos a criar usando uma instância como protótipo e cria novos objetos ao copiar este protótipo.
- **Singleton:** garante que uma classe só tenha uma única instância e provê um ponto de acesso global a ela.
- **Builder:** separa a construção de objeto complexo da representação para criar representações diferentes com o mesmo processo.
- **Abstract Factory:** provê interface para criar famílias de objetos relacionados ou dependentes, sem especificar suas classes concretas.

6. PADRÕES DE PROJETO GOF MAIS UTILIZADOS NO CONTEXTO DE JOGOS DIGITAIS:

Singleton

- Intenção: garantir que uma classe tem apenas uma instância e prover um ponto de acesso global a ela.

Solução: fazer com que a própria classe seja responsável pela manutenção da instância única, de tal forma que:

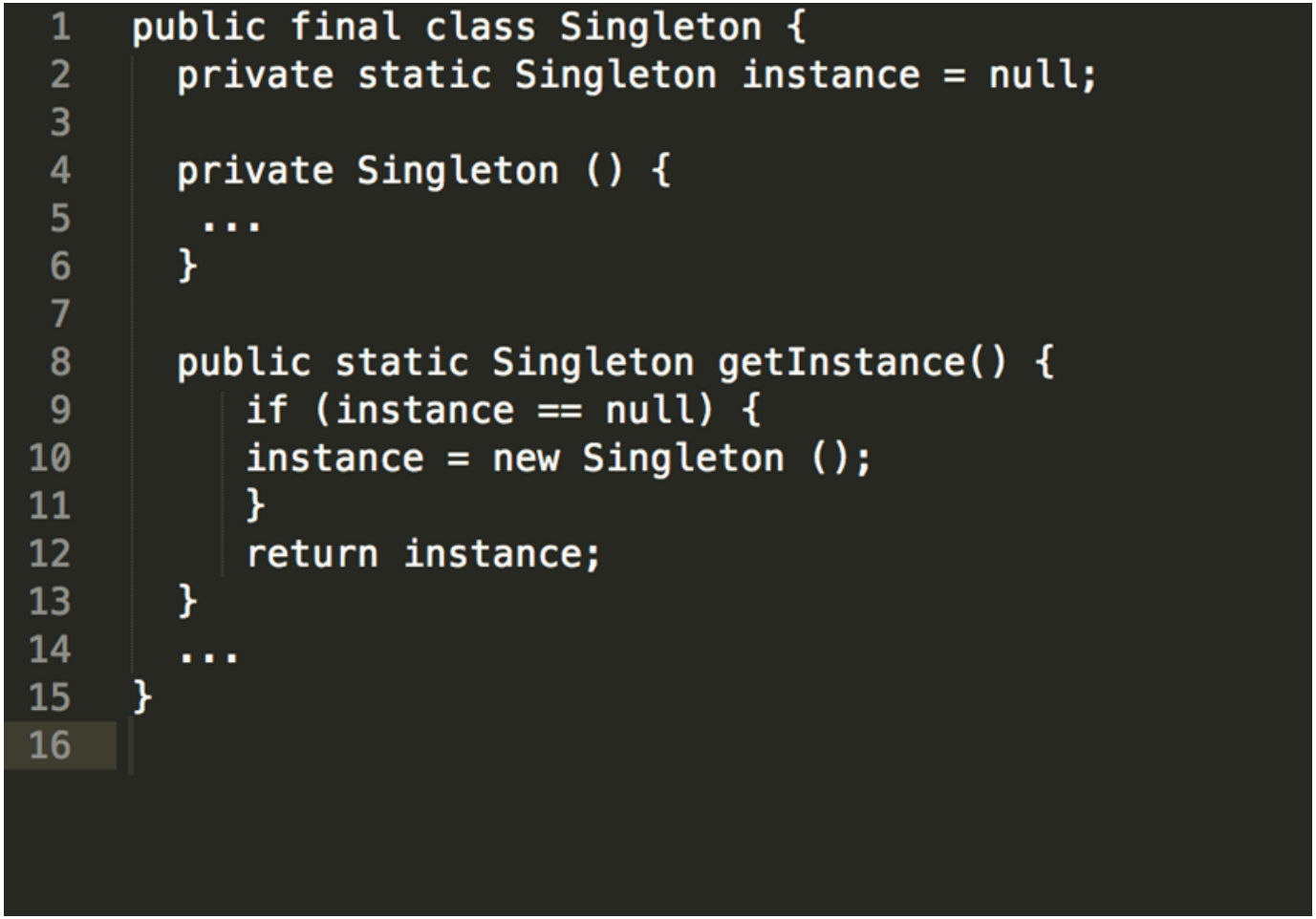
- Quando a instância for requisitada pela primeira vez, esta deve ser criada.
- Em requisições subsequentes, a instância criada na primeira vez é retornada.

A classe Singleton deve:

- armazenar a única instância existente;
- garantir que apenas uma instância seja criada;
- prover acesso a tal instância.

6.1. Implementação

Figura 6 – Implementação



```
1  public final class Singleton {
2    private static Singleton instance = null;
3
4    private Singleton () {
5     ...
6    }
7
8    public static Singleton getInstance() {
9       if (instance == null) {
10      instance = new Singleton ();
11      }
12      return instance;
13   }
14   ...
15 }
16
```

Fonte: Elaborada pelo autor.

6.2. Decorator

Problema:

- Adicionar para objetos individuais, dinâmica ou estaticamente, sem afetar o comportamento de outras classes e objetos.

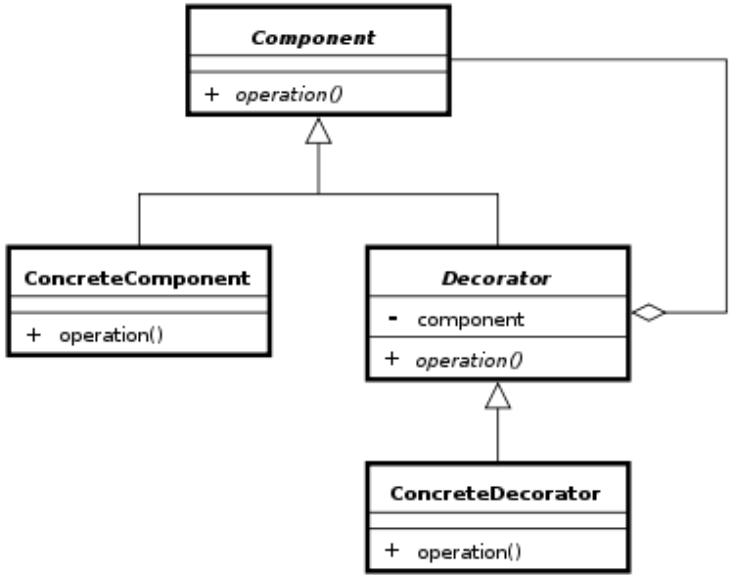
Motivação:

- Objeto possui funcionalidades básicas, mas é necessário adicionar funcionalidades a ele que podem ocorrer antes ou depois da funcionalidade básica.
- Funcionalidades são adicionadas em instâncias adicionais, e não na classe.

Aplicações:

- Você precisa usar a classe, e a interface não é equivalente ao que você precisa.
- Evitar explosões de subclasses.

Figura 7 – Decorator



Fonte: Elaborada pelo autor.

6.3. Observer

Problema:

- Define uma dependência um-para-muitos entre objetos de modo que, quando um objeto muda o estado, todos os seus dependentes são notificados e atualizados automaticamente.

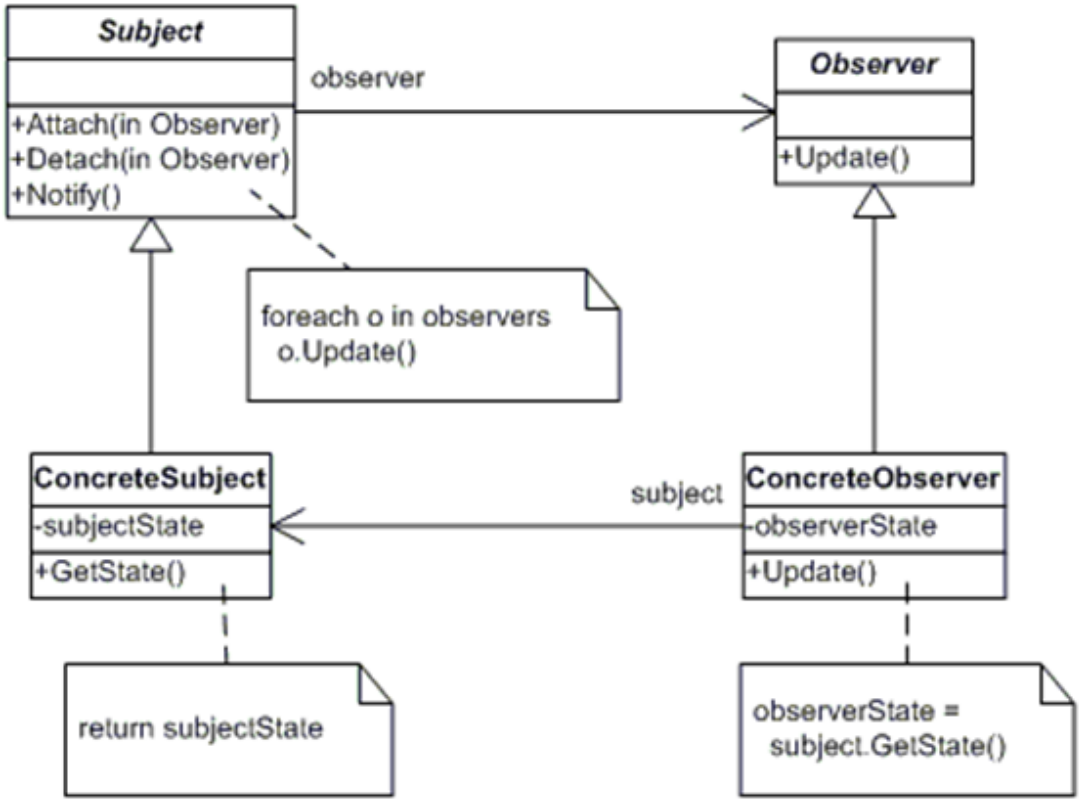
Motivação:

- Quando uma mudança em um objeto implica em uma mudança em outros.
- Quando um objeto deve poder notificar outros objetos sem assumir nada sobre eles.

Aplicações

- Broadcast.
- Publish/ Subscribe.
- Notificação de eventos.

Figura 8 – Observer



Fonte: GAMMA (1993).

7. REFERÊNCIAS

- GAMMA, E. et al. Design patterns: abstraction and reuse of object-oriented design. In: *European Conference on Object-Oriented Programming.* Berlin, Heidelberg, 1993, p. 406-431.
- PRESSMAN, R.; MAXIM, B. R. *Engenharia de software:* uma abordagem profissional. 8. ed. Porto Alegre: AMGH, 2016.