Padrão Estrutural

Eric Posser e Matheus Rossato

Introdução

São uma categoria de design patterns que se concentram na organização de classes e objetos para formar estruturas maiores, mantendo a flexibilidade e a facilidade de manutenção do código.

Esses padrões ajudam a definir como classes e objetos são compostos para formar estruturas complexas, facilitando a compreensão, a extensão e a modificação do código.

Eles são úteis quando se trata de relação entre entidades, organizando-as de maneira eficiente

Composite Pattern

Padrão Composite permite que objetos individuais e composições de objetos sejam tratados de maneira uniforme.

Ele define uma hierarquia de parte-todo, permitindo que clientes usem objetos individuais e composições de objetos de forma transparente.

Decorator Pattern

O padrão Decorator permite adicionar funcionalidades a objetos existentes de forma dinâmica.

Ele fornece uma maneira flexível de estender o comportamento de um objeto, sem modificar sua estrutura.

Facade Pattern

O padrão Facade fornece uma interface unificada para um conjunto de interfaces em um subsistema.

Ele define uma interface de nível mais alto que facilita o uso e a compreensão do subsistema, ocultando sua complexidade subjacente.

Problemas

Os padrões estruturais visam resolver problemas relacionados à organização e composição de classes e objetos em estruturas complexas

Esses problemas surgem em situações onde a estrutura do sistema precisa ser organizada de forma eficiente, mantendo a flexibilidade e a escalabilidade do código. Por exemplo, em sistemas com hierarquias de objetos complexos ou em subsistemas com interfaces intrincadas.

Soluções

Organizam classes e objetos de maneira flexível e eficiente. Eles definem relações claras entre entidades, simplificando a interação e promovendo a reutilização de código.

Os padrões estruturais geralmente incluem componentes como interfaces, classes base, operações….

Quando usar

São mais eficazes em cenários onde a estrutura do sistema é complexa e precisa ser organizada de forma clara e flexível.

Eles são adequados quando há a necessidade de criar hierarquias de objetos, adicionar funcionalidades a objetos existentes de forma dinâmica ou simplificar interfaces complexas de susbistemas.

Exemplo

Imagine um sistema de arquivos onde tanto arquivos individuais quanto diretórios(que pode conter arquivos e outros diretórios) precisam ser manipulados de maneira uniforme.

Neste exemplo, tanto os arquivos quanto os diretórios são tratados de maneira uniforme através da interface ‘FileSystemComponent’, permitindo que o cliente’’Main class’ liste o conteúdo de um diretório, independentemente de conter arquivos ou outros diretórios.

```java
interface FileSystemComponent {  5 usages  2 implementations
    void ls();  2 usages  2 implementations
}

class File implements FileSystemComponent {  4 usages

    public File(String name) {  2 usages
        this.name = name;
        private String name;
    }

    @Override  2 usages
    public void ls() {
        System.out.println("File: " + name);
    }
}

class Directory implements FileSystemComponent {  4 usages
    private String name;  2 usages
    private List<FileSystemComponent> children;  3 usages

    public Directory(String name) {  2 usages
        this.name = name;
        this.children = new ArrayList<>();
    }
```

```java
    public void add(FileSystemComponent component) {  3 usages
        children.add(component);
    }

    @Override  2 usages
    public void ls() {
        System.out.println("Directory: " + name);
        for (FileSystemComponent component : children) {
            component.ls();
        }
    }
}

public class Main {
    public static void main(String[] args) {
        Directory root = new Directory( name: "Root");
        Directory folder1 = new Directory( name: "Folder 1");
        File file1 = new File( name: "File 1");
        File file2 = new File( name: "File 2");

        root.add(folder1);
        folder1.add(file1);
        folder1.add(file2);

        root.ls();
    }
}
```

Vantagens e Desvantagens

Vantagens:

- Permite tratar objetos individuais e composições de objetos de maneira uniforme.
- Facilita a adição de novos tipos de elementos na estrutura hierárquica.
- Promove o reuso de código ao encapsular a estrutura hierárquica em componentes reutilizáveis.

Desvantagens:

- Pode ser difícil garantir que todos os componentes da hierarquia possuam suporte para as mesmas operações.

Conclusões

Padrões estruturais são ferramentas poderosas para organizar e simplificar a arquitetura de software.

Eles oferecem soluções para problemas comuns relacionados à composição e organização de classes e objetos, promovendo a flexibilidade, a reusabilidade e a manutenibilidade do código.

Ao entender e aplicar os padrões estruturais de forma adequada, os desenvolvedores podem construir sistemas mais robustos e escaláveis.