Conceitos

Design Patterns (Padrões de Projeto) são soluções típicas para problemas comuns em projetos de software. São como projetos de obras pré-fabricadas que podemos adaptar para resolver um problema de projeto recorrente no código.

Não é possível simplesmente encontrar um Design Pattern e copiá-lo para o programa. Não é um pedaço de código específico, apenas podemos seguir os detalhes.

Descrição

A maioria dos padrões são descritos formalmente para que as pessoas possam reproduzi-los em diferentes contextos.

- O propósito

Descreve brevemente o problema e a solução

- A Motivação

Explica a fundo o problema e a solução que o padrão torna possível

- As Estruturas

Mostram cada parte do padrão e como se relacionam

- Exemplos de código

Torna mais fácil compreender a ideia por trás do padrão

Alguns catálogos de padrão listam outros detalhes úteis, tais como a **aplicabilidade** do padrão, **etapas de implementação**, e relações com outros padrões.

Os Design Patterns são um kit de ferramentas para soluções tentadas e testadas para problemas comuns em projetos de software.

Design Patterns diferem em complexidade, nível de detalhe, e escala de aplicabilidade ao sistema inteiro que está a ser desenvolvido.

Os padrões mais básicos e de baixo nível são frequentemente chamados idiomáticos.

Os padrões mais universais e de alto nível são os padrões arquitetónicos.

- Os padrões criacionais (Creational Patterns) fornecem mecanismos de criação de objetos que aumentam a flexibilidade e a reutilização de código.
- Os padrões estruturais (Structural Patterns) explicam como instanciar objetos e classes em estruturas maiores, enquanto ainda mantém as estruturas flexíveis e eficientes.
- Os padrões comportamentais (Behavioral Patterns) cuidam da comunicação eficiente e da sinalização de responsabilidades entre objetos.

Singleton

O Singleton é um padrão de projeto criacional (Creational Pattern) que garante que uma classe tenha apenas uma instância, enquanto provê um ponto de acesso global para essa instância.

O padrão Singleton resolve dois problemas de uma só vez, violando o princípio de responsabilidade única:

1. Garantir que uma classe tenha apenas uma única instância. Porquè controlar quantas instâncias uma classe tem? A razão mais comum para isso é para controlar o acesso a algum recurso compartilhado— por exemplo, uma base de dados ou um arquivo.

2. Funcionamento: imagine que criou um objeto, mas passado algum tempo decidiu criar um novo. Ao contrário de receber um objeto criado de fresco, recebe um que já foi criado.

Fornece um ponto de acesso global para aquela instância. Podemos utilizar, para isso, variáveis globais... Embora sejam inseguras, pois a qualquer momento podem ser acedidas e modificadas. Assim como uma variável global, o padrão Singleton permite aceder um objeto a partir de qualquer lugar no programa. Contudo, ele também protege aquela instância de ser sobrescrita por outro código.

Todas as implementações do Singleton tem esses dois passos em comum:

1. Implementar o construtor padrão privado, para prevenir que outros objetos usem o operador new com a classe singleton.
2. Criar um método estático de criação que age como um construtor. Esse método chama o construtor privado, nos bastidores, para criar um objeto e salva num campo estático. Todas as invocações seguintes desse método retornam o objeto em cache.

Estrutura

- A classe Singleton declara o método estático getInstance que retorna a mesma instância da sua própria classe.
- O construtor da singleton deve ser escondido do código cliente. Chamando o método getInstance deve ser o único modo de obter o objeto singleton.

Utilize o padrão Singleton quando uma classe no programa deve ter apenas uma instância disponível para todos seus clientes.

Observe que podemos sempre ajustar essa limitação e permitir a criação de qualquer número de instâncias singleton. O único pedaço de código que requer mudanças é o corpo do método getInstance.

Uma classe fachada pode frequentemente ser transformada numa singleton já que um único objetofachada é suficiente na maioria dos casos. O Flyweight seria parecido com o Singleton se conseguissemos, de algum modo, reduzir todos os estados de objetos compartilhados para apenas um objeto flyweight. Mas há duas mudanças fundamentais entre esses padrões: Deve haver apenas uma única instância singleton, enquanto que uma classe flyweight pode ter múltiplas instâncias com diferentes estados intrínsecos. O objeto singleton pode ser mutável. Objetos flyweight são imutáveis. As Fábricas Abstratas, Construtores, e Protótipos podem todos ser implementados como Singletons.

Factory Method

O Factory Method é um padrão criacional de projeto que fornece uma interface para criar objetos numa superclasse, mas permite que as subclasses alterem o tipo de objetos que serão criados.

Solução

- O padrão Factory Method sugere que substitua chamadas diretas de construção de objetos (usando o operador new) por chamadas para um método fábrica especial.
- os objetos ainda são criados através do operador new, mas esse será invocado dentro do método fábrica. Objetos retornados por um método fábrica geralmente são chamados de produtos.
- As subclasses podem alterar a classe de objetos retornados pelo método fábrica.
- Porém, há uma pequena limitação: as subclasses só podem retornar tipos diferentes de produtos se esses produtos tiverem uma classe ou interface base em comum. Além disso, o método fábrica na classe base deve ter tipo de retorno declarado como essa interface.
- Todos os produtos devem seguir a mesma interface

Estrutura

1. O Produto declara a interface, que é comum a todos os objetos que podem ser produzidos pelo criador e suas subclasses.
2. Produtos Concretos são implementações diferentes da interface do produto. A classe Criador declara o método fábrica que retorna novos objetos produto.
3. É importante que o tipo de retorno desse método corresponda à interface do produto.
4. Criadores Concretos sobrescrevem o método fábrica base para retornar um tipo diferente de produto.

método fábrica base pode retornar algum tipo de produto padrão.

Aplicabilidade

- Utilize o Factory Method quando deseja fornecer aos utilizadores da sua biblioteca, framework ou programa uma maneira de estender seus componentes internos.
- Use o Factory Method quando deseja economizar recursos do sistema reutilizando objetos existentes em vez de recriá-los sempre.

Implementação

1. Faça todos os produtos implementarem a mesma interface. Essa interface deve declarar métodos que fazem sentido em todos os produtos.
2. Adicione um método fábrica vazio dentro da classe criadora. O tipo de retorno do método deve corresponder à interface comum do produto.
3. No código da classe criadora, encontre todas as referências aos construtores de produtos. Um por um, substitua-os por invocações ao método fábrica, enquanto extrai o código de criação do produto para o método fábrica.
4. Agora, crie um conjunto de subclasses criadoras para cada tipo de produto listado no método fábrica.
5. Se houver muitos tipos de produtos e não fizer sentido criar subclasses para todos, pode reutilizar o parâmetro de controlo da classe base nas subclasses.

6. Se, após todas as extrações, o método fábrica base ficar vazio, pode torná-lo abstrato. Se sobrar algo, pode tornar isso num comportamento padrão do método.

Strategy

O Strategy é um padrão de projeto comportamental (Behavioral Pattern) que permite definir uma família de algoritmos, colocá-los em classes separadas, e fazer os seus objetos intercambiáveis.

Solução

- O padrão Strategy sugere que pegue uma classe que faz algo específico de diversas maneiras diferentes e extraia todos esses algoritmos para classes separadas chamadas estratégias.
- A classe original, chamada contexto, deve ter um campo para armazenar uma referência para uma dessas estratégias. O contexto delega o trabalho para um objeto estratégia ao invés de executá-lo por conta própria.
- O contexto não é responsável por selecionar um algoritmo apropriado para o trabalho.

Estrutura

1. O Contexto mantém uma referência para uma das estratégias concretas e comunica com esse objeto através da interface da estratégia.
2. A interface Estratégia é comum a todas as estratégias concretas. Declara um método que o contexto usa para executar uma estratégia.
3. Estratégias Concretas implementam diferentes variações de um algoritmo que o contexto usa.
4. O contexto chama o método de execução no objeto estratégia ligado cada vez que ele precisa de executar um algoritmo. O contexto não sabe com que tipo de estratégia está a trabalhar ou como o algoritmo é executado.
5. O Cliente cria um objeto estratégia específico e passa como argumento para o contexto. O contexto expõe um setter que permite o cliente mudar a estratégia associada com contexto durante a execução.

Aplicabilidade

- Utilize o padrão Strategy quando quer usar diferentes variantes de um algoritmo dentro de um objeto e ser capaz de trocar de um algoritmo para outro durante a execução.
- Utilize o Strategy quando tem muitas classes parecidas que somente diferem na forma de executar algum comportamento.
- Utilize o padrão para isolar a lógica do negócio de uma classe dos detalhes de implementação de algoritmos que podem não ser tão importantes no contexto da lógica.
- Utilize o padrão quando a classe tem um operador condicional muito grande que troca entre diferentes variantes do mesmo algoritmo.

Implementação

1. Na classe contexto, identifique um algoritmo que é sujeito a frequentes mudanças. Pode ser também uma condicional enorme que seleciona e executa uma variante do mesmo algoritmo durante a execução do programa.
2. Declare a interface da estratégia comum para todas as variantes do algoritmo.
3. Um por um, extraia todos os algoritmos para as suas próprias classes. Elas devem todas implementar a interface estratégia.

4. Na classe contexto, adicione um campo para armazenar uma referência a um objeto estratégia. Forneça um setter para substituir valores daquele campo. O contexto deve trabalhar com o objeto estratégia somente através da interface estratégia. O contexto pode definir uma interface que deixa a estratégia aceder aos dados.
5. Os Clientes do contexto devem associá-lo com uma estratégia apropriada que coincide com a maneira que esperam que o contexto atue no seu trabalho primário.