Design Patterns

segunda-feira, 24 de janeiro de 2022 08:20

São Padrões de código para soluções de problemas conhecidos

O objetivo não é reinventar a roda e sim aplicar uma solução com um bom design de código

O conceito de padrões foi introduzido por 4 desenvolvedores intitulados Gang Of Four (GoF) e hoje conta com 23 padrões fundamentais

Atualmente existem mais de 80 padrões conhecidos que são em geral variações dos 23 Patterns do GoF

Os Padrões do GoF estão divididos em 3 famílias:

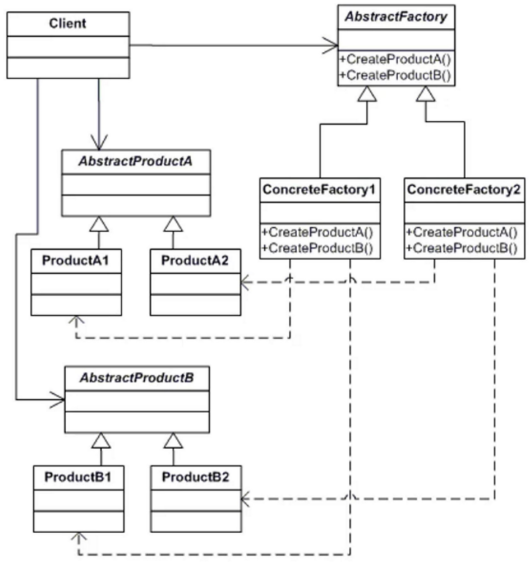
- Criacional
  Fornece meios de criação de um objeto e de como ele será instanciado
- Structural
  Tratam da composição de objetos por heranças e interfaces para diferentes funcionalidades
- Comportamental
  Tratam das interações e comunicação entre os objetos além da divisão de responsabilidades

Creational Patterns:

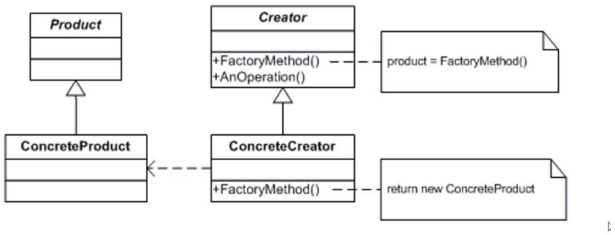
- Abstract Factory
  Cria uma instância de diversas famílias de classes
- Factory Method
  Cria uma instância de diversas derivações de classes
- Singleton
  Cria uma única instância que será utilizada por todos os recursos

C# Design Patterns - DoFactory

AbstractFactory

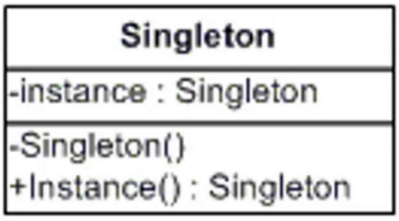


Factory Method -> Disponibiliza uma interface que irá te ajudar a criar um tipo especifico de produto,
A Classe que estiver consumindo irá tomar a decisão de qual produto irá criar



C	Abstract Factory	S	Facade	S	Proxy
S	Adapter	C	Factory Method	B	Observer
S	Bridge	S	Flyweight	C	Singleton
C	Builder	B	Interpreter	B	State
B	Chain of Responsibility	B	Iterator	B	Strategy
B	Command	B	Mediator	B	Template Method
S	Composite	B	Memento	B	Visitor
S	Decorator	C	Prototype		

As Letras e a cor são referente as famílias do Design Patterns

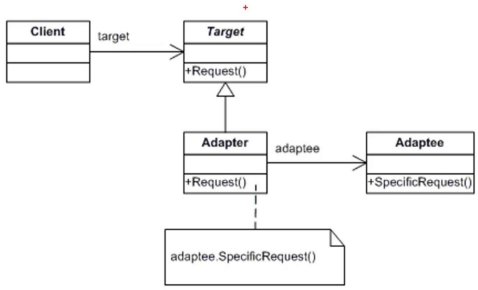


Instance => Propriedade privada que representa uma instância de um objeto
Instance() => Método que devolve umas instância deste objeto / retorna um instância ou se não houver cria

Structural Patterns

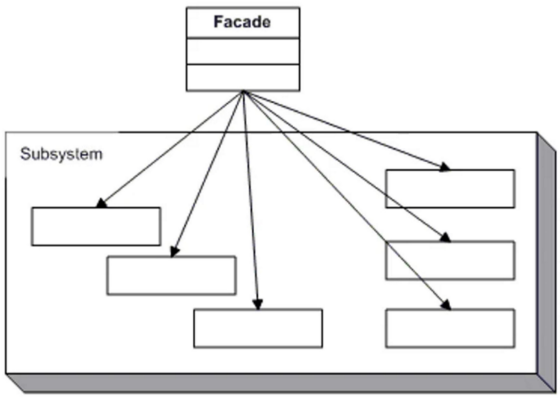
- Adptaer
  Compatibiliza objetos de interfaces diferentes
- Facade
  Uma única classe que representa um subsistema
- Composite
  Compartilha um objetoem estrutura de arvores que representam herarquias

Adpater



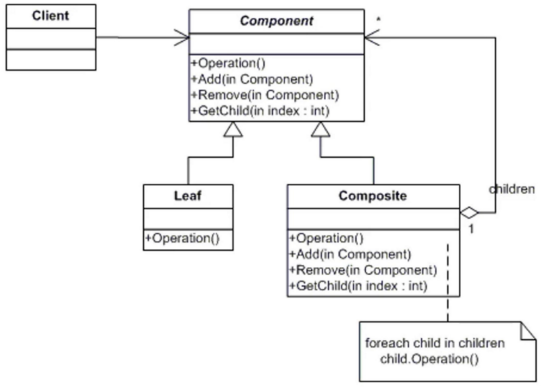
Target => Funcionalidade especifica que trabalha através de um interface implementada ou abstrata
Client => Está utilizando a funcionalidade Target
Adaptee => classe a ser adaptada para Target, porem elas não possuem a mesma interface que o cliente espera utilizar
Adapter => Criado este adpatar para substituir a interface do Target para Adaptee, para assim o cliente utilizar o objeto adaptador

Facade



Uma única classe que serve subsistema de uma parte da aplicação

Composite

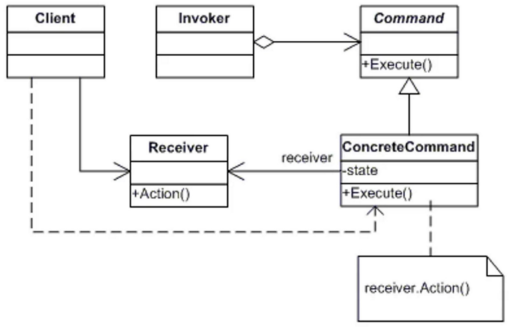


Gera uma hierarquia de classes do mesmo tipo

Behavior Patterns

- Command
  - Encapsula um command request em um objeto
- Strategy
  - Encapsula um algoritmo dentro de uma classe
- Observer
  - Uma forma de notificar mudanças a uma série de classes

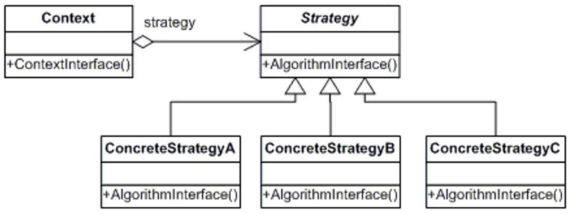
Command



Implementa um comando abstrato atraves de um command concreto

Receiver => é um lógica de alguma funcionalidade que será injetado nesse comando
Invoker => responsável por invocar o comando que invocara o command generico e receberá uma instância do command concreto

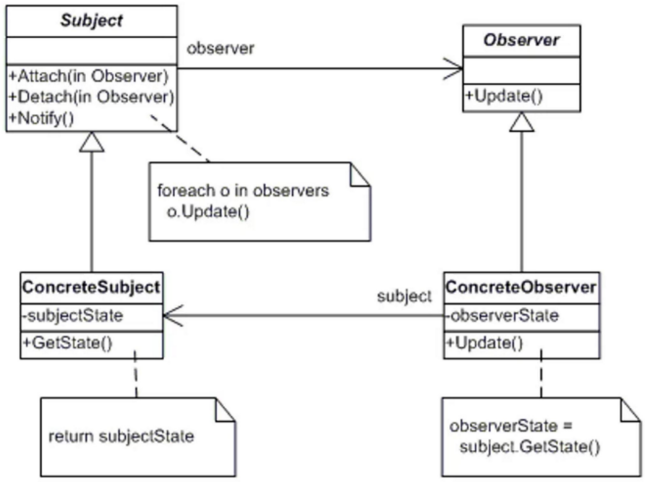
Strategy



Muito útil em modelagem de domínios, regras de negocio, a ideia é termos uma estratégia abstrata, na qual delegamos estratégias concretas, onde está sendo consumindo por um contexto, ex: Métodos de pagamento

Strategy e Facade trabalham muito bem juntos

Observer



Utilizado para fazer observações e notificações

Subject -> Classe abstrata
ConcreteSubject -> derivação concreta da Subject, Toda derivação concreta tem seu observador concreto
Do qual é derivado de um observer abstrato

Subject contem uma lista de observadores, todas vez que o Subject muda, a lista de observadores serão notificados