Design Patterns

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Ementa

- Introdução
- História
- Padrões Criacionais
- Padrões Estruturais
- Padrões Comportamentais

Introdução

Design Patterns

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Introdução

- Os padrões estão por toda parte! Na Arquitetura, por exemplo ajudam arquitetos a planejar prédios e discutir seus projetos. Na programação não é diferente, ajudam os programadores a **organizarem melhor seus programas** e **pensar mais sobre o código**.

- Existem diferentes áreas de programação onde padrões podem ser aplicados, desde aspectos organizacionais até na codificação.

- A partir de agora vamos tratar sobre os Design Patterns (Padrões de Projeto) – Padrões de Programação.

Introdução

- Padrão é algo que fizemos no passado, foi bem-sucedido e pode ser aplicado a várias situações. Os padrões capturam experiências no desenvolvimento de software que comprovadamente **funcionam repetidamente** e, assim, fornecem uma solução para problemas específicos. **Eles surgem da experiência prática.**

- Quando muitos programadores estão tentando resolver problemas semelhantes, eles sempre chegam a uma solução que funciona melhor. Essa solução é posteriormente destilada em um **modelo de solução**, algo que nós, programadores, usamos para abordar **problemas semelhantes** no futuro. Esses modelos de solução geralmente são chamados de padrões.

Introdução

- Bons padrões são **agnósticos** de problemas e de linguagem. Em outras palavras se aplicam em Java, C#, Delphi e etc...

- Muitos dos padrões, por exemplo, funcionam melhor com linguagens de programação orientada a objetos (POO).

- É importante observar que os padrões fornecem apenas um modelo para uma solução e não uma **receita detalhada**. Você ainda terá que cuidar do código e garantir que a sua implementação faça sentido e funcione no seu programa.

Introdução

- Os padrões são divididos em três grupos: **Criacionais**, **Estruturais** e **Comportamentais**.

- **Padrões de Projetos** x **Padrões Arquiteturais**

- **Padrões Arquiteturais:** “Um conjunto de princípios. Você pode pensar nisso como um padrão granular que oferece um framework abstrato para uma família de sistemas. Um padrão arquitetural melhora o particionamento e promove reuso de design fornecendo soluções para problemas recorrentes.”
- **Alguns Exemplos:** Client-Server, Multicamadas, Orientada a Objetos, Micros Serviços e etc...

História

Design Patterns

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

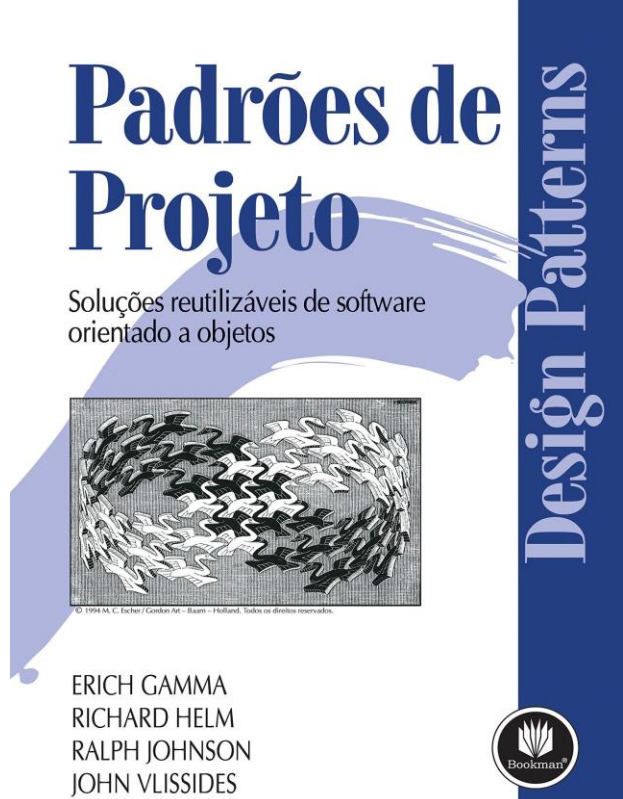
História

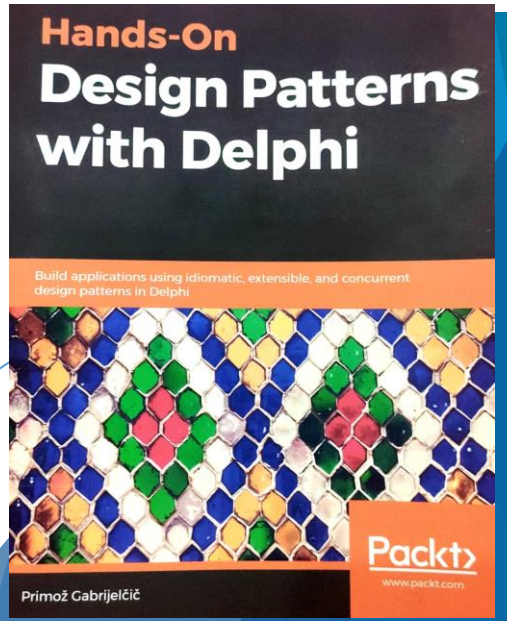
- **Gang of Four – GoF**

- O movimento do padrão de design foi declarado pela Gangue dos Quatro. São quatro autores de um livro denominado: “Design Patterns: Elements of Reusable Object-Oriented Software”.

- Este livro, escrito por **Erich Gamma**, **Richard Helm**, **Ralph Johson** e **John Vlissides**, foi publicado em 1994 e abalou completamente a comunidade de programação.

- A maior parte do livro é dedicada a 23 padrões de projeto de software. Esses padrões fornecem um núcleo do conjunto de ferramentas de um programador, embora, mais tarde, mais padrões tenham sido descobertos e formalizados.





23 Padrões de Projeto

Design Patterns

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Padrões Criacionais

CRIACIONAIS

- 1 – Abstract Factory
- 2 – Builder
- 3 – Factory Method
- 4 – Prototype
- 5 - Singleton

ESTRUTURAIS

- 6 – Adapter
- 7 – Bridge
- 8 – Composite
- 9 – Decorator
- 10 – Facade
- 11 – Flyweight
- 12 - Proxy

COMPORTAMENTAIS

- 13 – Chain of Responsibility
- 14 – Command
- 15 – Interpreter
- 16 – Iterator
- 17 – Mediator
- 18 – Memento
- 19 – Observer
- 20 – State
- 21 – Strategy
- 22 – Template Method
- 23 - Visitor

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Padrões Criacionais

Design Patterns

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Padrões Criacionais

Primeiro grupo dos padrões, são usados quando objetos simples e compostos são criados.

- 1 – Abstract Factory
- 2 – Builder
- 3 – Factory Method
- 4 – Prototype
- 5 - Singleton

Padrões Criacionais – Abstract Factory

Declara uma interface responsável por criar uma fábrica de objetos similares sem especificar suas implementações concretas.

- Eliminação de “IFs” no seu código.
- Mantém a escalabilidade do seu código.

```
procedure MostrarDadosProdutos;
begin
  // Dados do notebook
  if Opcao = 'Dell' then
  begin
    EditTamanhoTela.Text := 'Tela de 14 polegadas';
    EditMemoriaRAM.Text := '3GB DDR3';
  end
  else if Opcao = 'Apple' then
  begin
    EditTamanhoTela.Text := '11.6 polegadas';
    EditMemoriaRAM.Text := '4GB DDR3';
  end;

  // Dados do desktop
  if Opcao = 'Dell' then
  begin
    EditProcessador.Text := 'Intel Core i5';
    EditTamamhoHD.Text := '1 TB';
  end
  else if Opcao = 'Apple' then
  begin
    EditProcessador.Text := 'Intel Core i7';
    EditTamamhoHD.Text := '500 GB';
  end;
end;
```

```
procedure MostrarProdutos;
var
  Marca: IFactoryMarca;
  Notebook: INotebook;
  Desktop: IDesktop;
begin
  // instancia a marca -> único IF da aplicação
  if Opcao = 'Dell' then
    Marca := TDell.Create
  else if Opcao = 'Apple' then
    Marca := TApple.Create;

  // consulta (constrói) os objetos
  Notebook := Marca.ConsultarNotebook;
  Desktop := Marca.ConsultarDesktop;

  // exibe os dados
  EditTamanhoTela.Text := Notebook.BuscarTamanhoTela;
  EditMemoriaRAM.Text := Notebook.BuscarMemoriaRAM;
  EditProcessador.Text := Desktop.BuscarNomeProcessador;
  EditTamamhoHD.Text := Desktop.BuscarTamanhoHD;
end;
```

Padrões Criacionais – Abstract Factory

Exemplo na prática: Consulta de Produtos

- O cliente precisa informar a marca e visualizar a descrição da especificações do notebook e desktop daquela marca.

Padrões Criacionais – Builder

Permite encapsular a lógica de construção de objetos, de modo que diferentes representações possam ser construídas com a mesma Interface.

- Existem quatro elementos neste padrão: **Director**, **Builder**, **Concrete Builder** e **Product.**

- **Exemplo**: Em um fast-food você realiza o pedido com um atendente. Este, por sua vez, repassa o pedido para a cozinha, onde o seu pedido é preparado. Quando pronto, o lanche é entregue ao atendente e, por fim, a você.

Padrões Criacionais – Builder

Observa-se:

- O atendente é apenas um intermediário entre o cliente e a cozinha.
- O atendente não conhece como o lanche é feito. Apenas repassa o pedido do cliente.
- O lanche é feito pelos funcionários da cozinha, enquanto o atendente apenas espera ficar pronto.

Termos Técnicos:

- O atendente é o **Director**, que solicita a construção de um objeto (Produto).
- O lanche é o **Product**, ou seja, o produto resultante da construção.
- O quadro de funcionários que fazem os lanches é o **Builder** (Interface)
- O funcionário que preparou o lanche é o **Concrete Builder** (Objeto).

Padrões Criacionais – Builder

Exemplo na prática – Construção de Relatórios

- Construção do cabeçalho
- Construção do corpo
- Construção do rodapé

Padrões Criacionais – Factory Method

Disponibiliza uma forma de reduzir a complexidade ciclomática ao delegar a lógica de criação de objetos para as classes derivadas.

- “A principal diferença entre Factory Method e Abstract Factory é que o primeiro representa um método simples, enquanto o segundo é um objeto. Por ser um método, o Factory Method pode ser sobrescrito em subclasses. O Abstract Factory, por sua vez, é um objeto que pode conter várias fábricas.”

- O Factory Method é um padrão de projeto que encapsula a criação de um objeto através de um método (por isso o “method” no nome). Por meio de parâmetros, a fábrica (factory) decide qual produto (product) deve criar e retornar para o cliente.

Padrões Criacionais – Factory Method

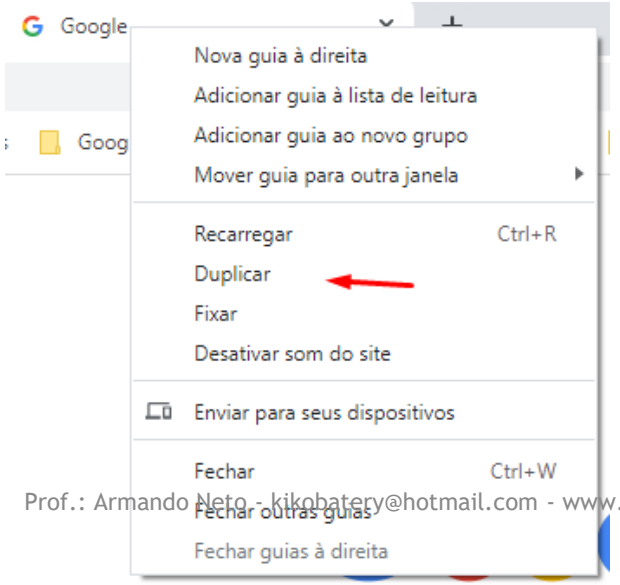
Exemplo na prática: Financiadora

- O cliente informará o valor, a quantidade de parcelas e o prazo de pagamento deste valor. O **Factory Method** irá criar um objeto de tipo de prazo com todas as informações geradas com estes parâmetros, como o número da parcela, a data de vencimento, o valor atualizado e o valor total a pagar.

Padrões Criacionais – Prototype

Permite clonar o estado e as propriedades de um objeto já existente para um novo objeto, criando uma cópia idêntica.

- O Prototype é geralmente utilizado para criar um clone de um objeto, de forma que este possa receber novos valores para os atributos sem impactar o objeto original.



Padrões Criacionais – Prototype

Exemplo na prática: Clonar Reuniões

- Possibilitar o usuário de clonar a reunião, evitando que tenha retrabalho em preencher as informações tudo novamente.

Padrões Criacionais – Singleton

Provê um ponto único de acesso à instância de um objeto, de modo que qualquer local da aplicação consiga acessá-lo.

- Assemelha-se também como “objeto global’ da aplicação.
- Exemplo: Geralmente, em sistemas multiusuários, é comum buscar as permissões de acesso e outros dados do usuário (customizações, padrões, temas e etc...) para cada tela acessada. Pensando na programação, cria-se um objeto da classe, faz a leitura dos dados solicitados, e o destrói em seguida – **Para evitar essa redundância de criação podemos usar o padrão Singleton**
- **Cuidado:** Ao utilizá-lo abusivamente, torna-se um anti-padrão. Além do objeto ficar na memória se criarmos vários reduziremos o desempenho da nossa aplicação.
- **Lembre-se:** Vai usar Singleton? Pense três vezes antes de utilizá-lo.

Padrões Criacionais – Singleton

Exemplo na prática: Logs

- Registrar logs em diferentes partes do sistema.

Padrões Estruturais

Design Patterns

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Padrões Estruturais

Manipula a maneira como os objetos são compostos em novos objetos.

- 6 – Adapter
- 7 – Bridge
- 8 – Composite
- 9 – Decorator
- 10 – Facade
- 11 – Flyweight
- 12 - Proxy

Padrões Estruturais – Adapter

Atua como intermediário entre duas interfaces, de modo que elas possam se comunicar, tornando-se compatíveis.

Padrões Estruturais – Adapter

Embora o Adapter possa ser empregado em várias situações, as mais comuns são:

- Integração com tecnologias externas, como WebServices para consultas de dados, pagamentos ou envio de arquivos.
- Integração entre sistemas diferentes, como aplicações desktop com aplicações web ou mobile.
- Migração de dados, quando, por exemplo, uma empresa adquire outra do mesmo segmento e pretende unificar as bases de dados.

Padrões Estruturais – Adapter

No detalhamento técnico, o Adapter é formado por 4 elementos:

- **Client:** Cliente que irá consumir os objetos
- **Target:** Define a interface que o Client usará
- **Adaptee:** Consiste na classe que precisa ser adaptada
- **Adapter:** Realiza a adaptação entre as interfaces

Padrões Estruturais – Adapter

Exemplo na prática: Busca de Endereços

- Vamos consultar os endereços a partir do CEP, usando os serviços da ViaCep e dos Correios.

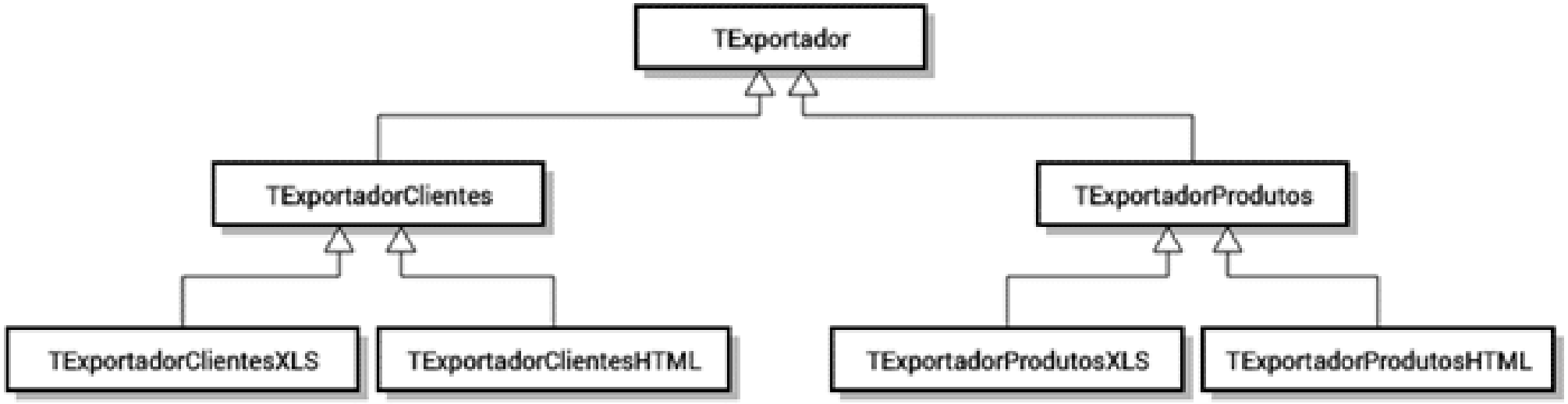
Padrões Estruturais – Bridge

Elimina múltiplas heranças e reduz a quantidade de classes ao desacoplar abstrações de suas implementações.

- Principal objetivo é **eliminar múltiplas heranças** e reduzir a quantidade de classes existentes no projeto.

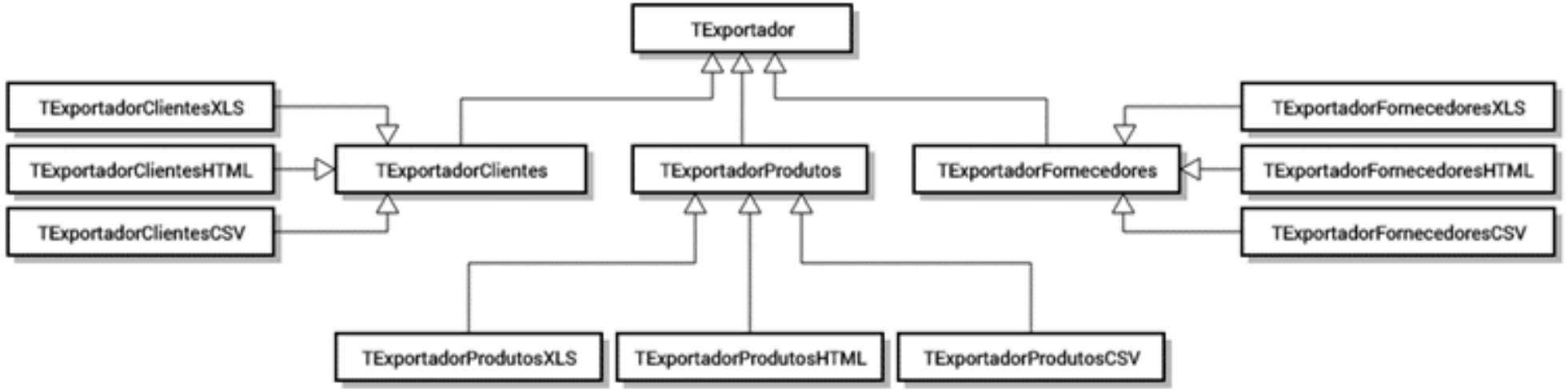
- **Exemplo:** Vamos imaginar uma aplicação que exporta os dados de clientes e produtos em formatos XLS e HTML. Segue o diagrama a seguir...

Padrões Estruturais – Bridge



Agora vamos supor que o usuário te solicitou a exportação também dos fornecedores e também que todos possam dados possam ser exportados em um novo formato, o CSV. E agora? Segue o diagrama...

Padrões Estruturais – Bridge



Foram criadas mais 6 novas classes (4 para os fornecedores e 2 para o novo formato de exportação para as classes de Produtos e Clientes).

Padrões Estruturais – Bridge

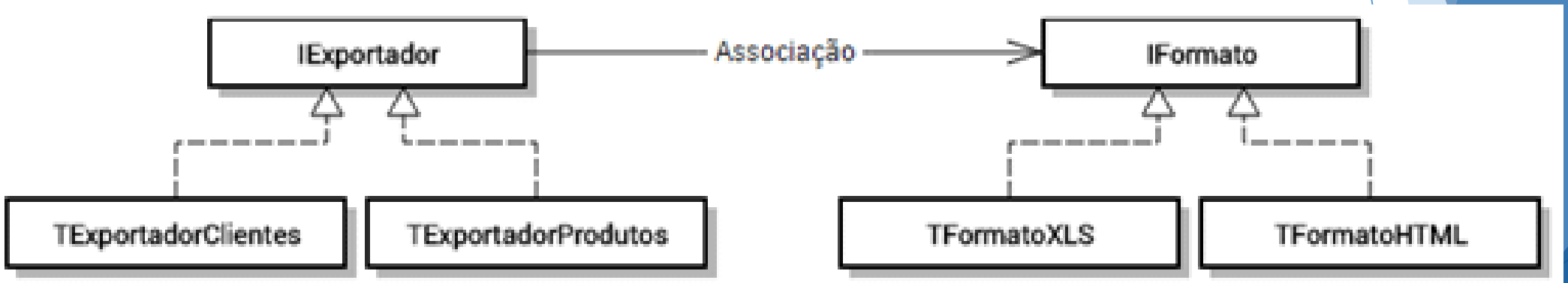
Como melhorar? Usando o padrão Bridge para **separar e agrupar as responsabilidades.**

Este padrão é composto por 4 elementos:

- **Abstraction:** Classe abstrata ou interface da abstração principal.
- **Refined Abstraction:** Implementação concreta da Abstraction.
- **Implementor:** Interface da abstração utilizada pela Abstraction.
- **Concrete Implementor:** Implementação concreta da Implementor.

Padrões Estruturais – Bridge

Então como ficaria a arquitetura do exemplo usando o Bridge?



Padrões Estruturais – Composite

Proporciona uma técnica para trabalhar com objetos individuais e grupos de objetos (composição) de modo uniforme.

- Fazendo uma analogia de ferramentas (objeto “parte”) e caixa de ferramentas (objeto “todo”). Para algumas operações, precisamos apenas de uma ferramenta específica, como uma chave de fenda. Em outras ocasiões, como o conserto de um veiculo, é necessário utilizar um conjunto, ou melhor, uma composição de ferramentas.
- **Exemplo:** Em uma esteira de produção cada máquina é responsável por realizar o seu trabalho.

Padrões Estruturais – Composite

Nesse padrão existem 4 elementos:

- **Component:** Representa uma interface que será implementada pelas classes relacionadas à composição, ou seja, pela classe principal e suas “ramificações”.
- **Operation:** Método comum entre esses objetos que a interface acima terá.
- **Composite:** Classe concreta de composição que possui uma lista de objetos (filhos), como se fosse uma árvore.
- **Leaf:** São as classes filhas, ou “folhas” da árvore, que estarão atreladas à uma composição.

Padrões Estruturais – Composite

Exemplo na prática: Viagens e Pacotes de Viagens

- Um cliente poderá realizar uma única viagem como também pode optar por montar um pacote de viagens, havendo assim uma consulta de valor.

Padrões Estruturais – Decorator

Visa adicionar comportamentos extras a objetos em tempo de execução, reduzindo a necessidade de criar heranças.

- O objetivo do Decorator é de **adicionar funcionalidades extras a um objeto em tempo de execução**, de forma que ele possa ter novos comportamentos que originalmente não fazem parte da sua estrutura.

Padrões Estruturais – Decorator

Este padrão também possui 4 elementos:

- **Component:** Interface comum que será implementada tanto pelas classes que poderão ser decoradas quanto pelas classes decoradoras.
- **Concrete Component:** Classes que implementam Component e que apenas poderão receber “decorações”, ou melhor, receber funcionalidades extras.
- **Decorator:** Classe abstrata que implementa Component e atua como classe base de todas as decorações possíveis.
- **Concrete Decorator:** Classes que herdam de Decorator e exercem o papel de “decoradoras”

Padrões Estruturais – Decorator

Exemplo na Prática: Exibindo exceções do sistema.

- Vamos praticar o Decorator com um exemplo de exceções que ocorrem no nosso sistema.

Padrões Estruturais – Facade

Disponibiliza um acesso simplificado a um conjunto de procedimentos complexos, também definido como “subsistemas”

- O propósito do Facade é encapsular um procedimento complexo, que geralmente envolve uma biblioteca de classes, e disponibilizar ao cliente apenas um método simplificado para executá-lo.
- A tradução, “Fachada”, é uma analogia para indicar o que o cliente pode acessar externamente, poupando-o de conhecer detalhes complexos internos.
- **Exemplo:** Seria fazermos uma analogia com uma TV o botão liga/desliga é uma fachada onde internamente são executadas várias operações internas.

Padrões Estruturais – Facade

Exemplo na Prática: Cálculo de Valor de Venda A Partir do Dólar.

- Vamos consumir um WebService para realizar a consulta do valor do Dólar do dia e realizaremos o cálculo do valor de venda de acordo com a margem de venda.

Padrões Estruturais – Flyweight

Aprimora o desempenho e consumo de memória de uma aplicação através do compartilhamento de objetos em tempo de execução.

- O seu principal objetivo é melhorar o desempenho de um procedimento através de **compartilhamento de objetos com características similares**. Em outras palavras, prevê um mecanismo para utilizar objetos já existentes, modificando algumas propriedades, ao invés de ficar instanciando novos objetos.
- Segue o mesmo propósito do padrão Singleton, porém a diferente que o Flyweight trabalha com vários objetos de uma única vez.
- Os objetos desse padrão devem possuir dois tipos de propriedades:
- Intrínsecas: Propriedades imutáveis, ou seja, que caracterizam o objeto compartilhado.
- Extrínsecas: Propriedades variáveis que podem receber novos valores a cada acesso.

Padrões Estruturais – Flyweight

Exemplo na Prática: Lote de Cartões de Agradecimentos

- Para praticar imagina o cenário onde o usuário tem que enviar cartões personalizados de acordo com o país e o nome do cliente.

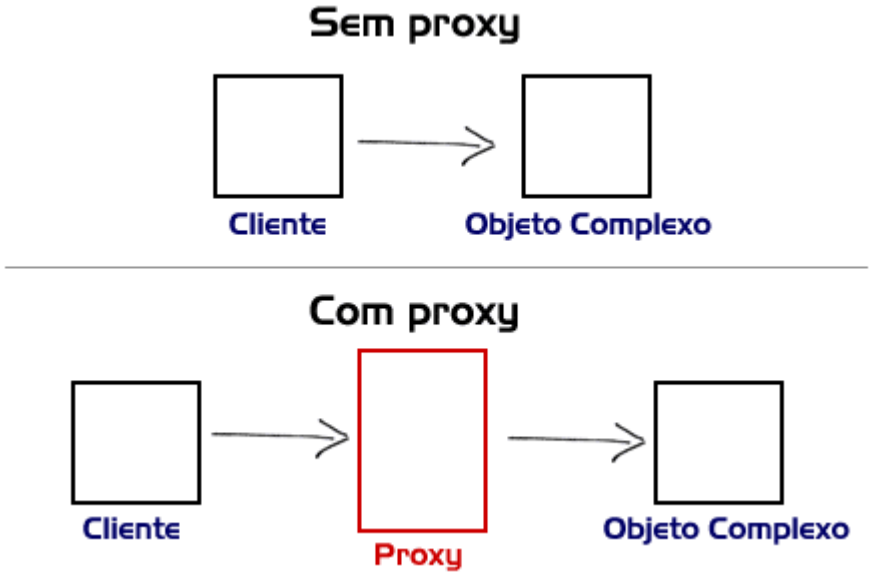
Padrões Estruturais – Proxy

Declara uma classe que controla o acesso e utilização de um objeto complexo ou protegido, aprimorando a segurança e/ou performance.

- Este padrão atua como um intermediário entre dois lados, é capaz de aperfeiçoar o desempenho de uma rotina em uma aplicação, executando validações e tratando dados, por exemplo, de forma que contribua para essa finalidade.
- O Proxy é adequado para cenários em que o cliente utiliza um objeto de uma **classe extensa ou complexa**, na qual consome muita memória ou afeta o desempenho da aplicação.

Padrões Estruturais – Proxy

Ilustrando...



Padrões Estruturais – Proxy

Exemplo: Fechando de uma Caixa de Supermercado

```
var
  ObjetoComplexo: TObjetoComplexo;
begin
  // Cria o objeto complexo, consumindo memória
  ObjetoComplexo := TObjetoComplexo.Create;

  if not ObjetoComplexo.VerificarPermissao then
    Exit;

  if not ObjetoComplexo.VerificarTodasInstanciasEstaoFechadas then
    Exit;

  // O caixa será fechado somente se passar nas duas validações acima
  ObjetoComplexo.FecharCaixa;
end;
```

```
procedure TProxy.FecharCaixa;
var
  ObjetoComplexo: TObjetoComplexo;
begin
  // método interno do proxy
  if not VerificarPermissao then
    Exit;

  // método interno do proxy
  if not VerificarTodasInstanciasEstaoFechadas then
    Exit;

  // Cria o objeto complexo somente se as validações forem satisfeitas
  ObjetoComplexo := TObjetoComplexo.Create;

  ObjetoComplexo.FecharCaixa;
end;
```

```
var
  Proxy: TProxy;
begin
  Proxy := TProxy.Create;
  Proxy.FecharCaixa;
end;
```

Padrões Estruturais – Proxy

Exemplo na Prática: Cálculo de Distancia entre duas Cidades

- Para praticar vamos fazer um sistema que calcule a distancia entre duas cidades, utilizando a API do Google.

Padrões Comportamentais

Design Patterns

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Padrões Comportamentais

Lida com a comunicação entre objetos e com a distribuição de responsabilidades

- 13 – Chain of Responsibility
- 14 – Command
- 15 – Interpreter
- 16 – Iterator
- 17 – Mediator
- 18 – Memento
- 19 – Observer
- 20 – State
- 21 – Strategy
- 22 – Template Method
- 23 - Visitor

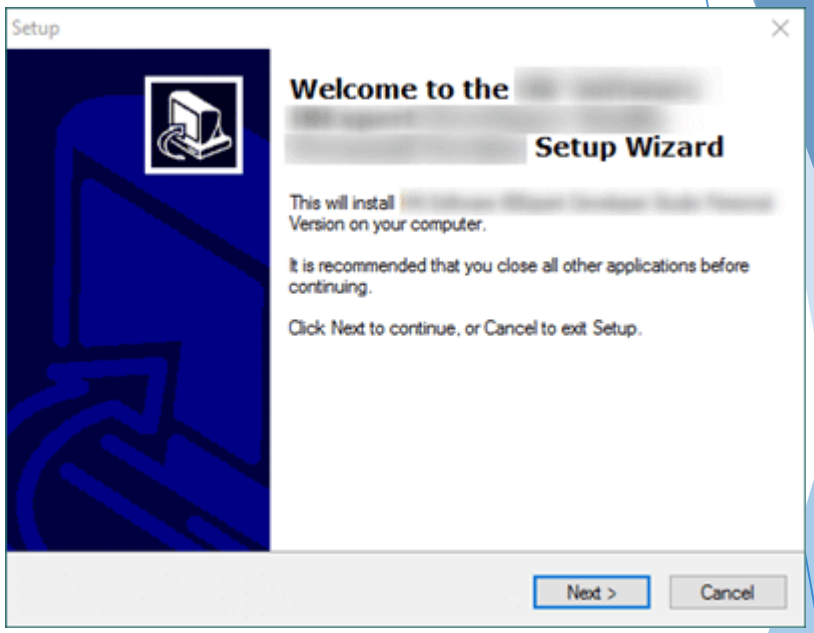
Padrões Estruturais – Command

Visa encapsular requisições como objetos e permite a criação de filas de execução de comandos de forma customizada.

- O objetivo do Command **é encapsular requisições como objetos**, possibilitando a criação de filas de comandos a serem executados sob determinada ordem. Log, podemos “empilhar” vários comandos e executá-los de uma só vez somente quando necessário.

Padrões Estruturais – Command

Exemplo: Next, Next, Next e Finish



- O Wizard dos instaladores é um ótimo exemplo, ele “empilha” os comandos que foram selecionados em cada etapa do next para que no finish sejam executados.

```
Instalador.Add(ComandoConfigurarPermissoes);
Instalador.Add(ComandoCopiarArquivos);
Instalador.Add(ComandoCriarAtalhos);

Instalador.InstalarPrograma;
```

Padrões Estruturais – Command

Exemplo na Prática: Extrator de informações do computador.

- Vamos criar um exemplo na prática de um extrator de informações do computador, listando os processos em execução, programas instalados e as variáveis de ambiente.

Padrões Estruturais – Observer

Um objeto específico é observado por outros objetos, de modo que estes sejam automaticamente notificados quando uma alteração ocorre no objeto principal.

- **Exemplo:** É o funcionamento do Windows Explorer. Se você abrir três janelas do mesmo diretório e criar um arquivo em uma delas as demais janelas serão atualizadas. Isso ocorre porque o diretório é o objeto **"observável"**, enquanto as janelas são os objetos **"observadores"**.

Este padrão possui 4 elementos:

- **Subject:** Interface que define a assinatura de métodos das classes que serão observáveis.
- **Concrete Subject:** Implementação da Interface Subject.
- **Observer:** Interface que define a assinatura de métodos das classes que serão observadoras.
- **Concrete Observer:** Implementação da Interface Observer.

Padrões Estruturais – Observer

Exemplo na Prática: Controle Financeiro

- O sistema deverá atualizar os dados financeiros em painéis distintos: Balanço, Valor de Débitos e Log do histórico de operações.

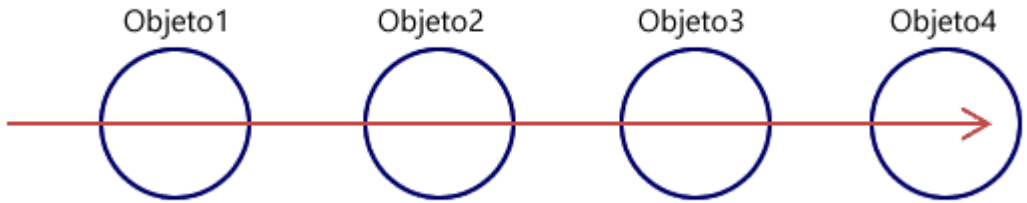
Padrões Estruturais – Chain of Responsability

Consiste em criar uma cadeia de objetos processadores, no qual cada um é responsável por processar um tipo de mensagem específica.

- Este padrão representa um solução para a redução de dependências entre classes. O maior propósito é permitir que mensagens (ou dados) naveguem entre diferentes objetos de uma hierarquia (ou cadeia) até que um desses tenha a capacidade de assumi-la, ou melhor, processá-la, mas com um detalhe importante: **cada objeto não conhece os detalhes do outro (alta coesão e baixo acoplamento).**

Padrões Estruturais – Chain of Responsability

- **Exemplo:** Aprovação multinível de um orçamento. Neste exemplo, o orçamento, que é a “mensagem”, é enviado primeiro para o Gerente, que possui permissão para aprovar 10 mil reais. Se o valor for maior, até 20 mil reais, é enviado o próximo cargo da hierarquia que, neste caso, é o diretor. E caso o valor seja superior que 20 mil reais, o orçamento é enviado ao CEO.



- Se o segundo objeto for removido, o sucessor do 1º elo passará a ser o 3º. A mensagem, por sua vez, não sofre impacto com a alteração da corrente sendo transmitida naturalmente pela hierarquia até que seja processada por algum objeto.

Padrões Estruturais – Chain of Responsability

Exemplo na Prática: Importação de Arquivos.

- Vamos criar uma aplicação para importar os dados de diferentes formatos de arquivos (.csv, .xml e .json). Cada objeto deverá identificar se tem capacidade para processar senão passar ao sucessor.

Padrões Estruturais – Interpreter

Define uma representação para uma linguagem e fornece procedimentos para interpretá-la dinamicamente.

- **Exemplo:** O Google Tradutor é um exemplo, o serviço é capaz de interpretar o texto digitado pelo usuário e traduzi-lo para o idioma desejado.

- Outro exemplo é a conversão de algarismos romanos em números decimais.

Padrões Estruturais – Interpreter

Exemplo na Prática: Traduzir frases em instruções SQL

- Vamos criar um interpretador de frases para comandos SQL, por exemplo: Atualizar o nome do cliente 2 para Armando Neto -> Update clientes set nome = ‘Armando Neto’ Where ID = 2.

Padrões Estruturais – Iterator

Consiste em um mecanismo para navegar entre os elementos de uma lista sem expor a representação dos dados processados.

- Com o *Iterator* podemos “transformar” uma classe em uma estrutura que permita a iteração dos dados processados por ela.

- A proposta deste padrão é disponibilizar uma forma de percorrer uma coleção (lista) sem a necessidade de conhecer a representação dos dados.

Padrões Estruturais – Iterator

Este padrão possui 4 elementos:

- **Iterator:** Define o contrato dos métodos de navegação.
- **Aggretate:** Define um método para a criação do Iterator.
- **Concrete Iterator:** Define a codificação dos métodos de navegação.
- **Concrete Aggregate:** Responsável pela criação do Iterator, informando a lista que será manipulada.

Padrões Estruturais – Iterator

Exemplo na Prática: Importador de Dados.

- Vamos verificar na prática o Iterator com um aplicativo de importação de dados.

Padrões Estruturais – Mediator

Encapsula a interação entre objetos para reduzir a dependência, promovendo o baixo acoplamento na arquitetura.

- O *Mediator* provê uma forma de encapsular as interações entre diferentes objetos. Ou seja, quando duas classes precisam se comunicar, essa comunicação é feita pelo *Mediator*.
- O maior objetivo é diminuir o nível de acoplamento já que as classes não vão conhecer umas as outras.
- **Exemplo:** Controle de tráfego aéreo. Os aviões não "conhecem" uns aos outros, porém conseguem se comunicar para não gerarem acidentes. Essa comunicação e administração é feita pela torre de controle que atua como um *Mediator*.

Padrões Estruturais – Mediator

Este padrão possui 4 elementos:

- **Mediator:** Interface que define os métodos de comunicação entre os objetos.
- **Concrete Mediator:** Classe concreta que implementa a Interface Mediator.
- **Colleague:** Interface que define os métodos referentes às “Intenções” dos objetos.
- **Concrete Colleague:** Classe concreta que implementa a Interface Colleague.

Padrões Estruturais – Mediator

Exemplo na Prática: Aplicação de Compra e Venda

- Desenvolveremos uma aplicação de Compra e Venda onde os membros enviarão propostas para outros membros mesmos não os conhecendo.

Padrões Estruturais – Memento

Cria um recurso de armazenamento de estados de um objeto, permitindo suas restaurações quando necessário.

- O objetivo do *Memento* é restaurar o estado anterior do objeto. Semelhante quando realizamos o Rollback na transação do banco de dados, ou quando usamos o CTRL+Z.

Elementos presentes neste padrão:

- **Memento:** Classe que armazena os atributos que poderão ser restaurados.
- **Originator:** Tem a função de "Originar" os estados do objeto.
- **Caretaker:** Classe para armazenar a lista de restaurações.

Padrões Estruturais – Memento

Exemplo na Prática: Editor de Texto.

- Vamos criar um editor de texto para salvar o histórico de alterações realizadas.

Padrões Estruturais – State

Permite que um objeto modifique o seu comportamento de acordo com o seu estado, proporcionando um alto nível de encapsulamento.

- O padrão State serve para quando a regra de negócio da sua aplicação mudar o comportamento do seu objeto também mudar.

```
if ObjetoCliente.Situacao = 'A' then // Ativo
  AplicarJurosClienteAtivo
else if ObjetoCliente.Situacao = 'P' then // Com Pendências
  AplicarJurosClientePendente
else if ObjetoCliente.Situacao = 'B' then // Bloqueado
  AplicarJurosClienteBloqueado;
```

```
ObjetoCliente.AplicarJuros;
```

Padrões Estruturais – State

Este padrão possui 3 elementos:

- **State:** Interface que declara os métodos que poderão ser executados pelos estados do objeto.
- **Concrete State:** Implementa a Interface State para definir cada estado possível do objeto.
- **Context:** Representa o estado atual do objeto, invocando seus métodos.

Padrões Estruturais – State

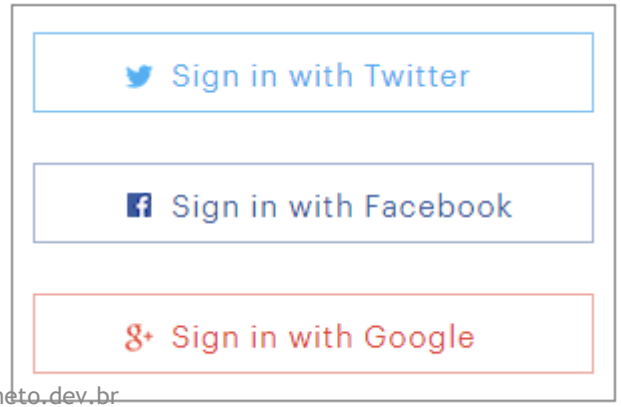
Exemplo na Prática: Aplicação para Pedidos.

- Criaremos uma solução para pedidos e de acordo com o valor total do pedido vamos classificar seu pedido.

Padrões Estruturais – Strategy

Fornece a capacidade de selecionar um algoritmo em tempo de execução, conforme a situação atual (ou regras de negócio) para executar uma tarefa específica.

- O objetivo desse padrão é que a sua aplicação possa selecionar e usar um algoritmo em tempo de execução, de acordo com as condições da sua regra de negócio.
- **Exemplo:** Autenticação na aplicação utilizando contas de terceiros (Google, Facebook e etc...).



Padrões Estruturais – Strategy

- Outro exemplo seria a forma de pagamento de uma compra (Crédito, Débito ou Voucher), cada uma tem uma estratégia, porém com o mesmo objetivo, liquidar a compra.

Exemplo na Prática: Validador de e-mail

- Vamos desenvolver um validador de e-mail onde o usuário poderá escolher qual algoritmo ele quer usar.

Padrões Estruturais – Template Method

Estabelece uma sequencia de passos e permite que alguns deles sejam “delegados” para as subclasses em tempo de execução.

- Este padrão estabelece uma sequencia de passos, porém, alguns deles são “delegados” para as subclasses em tempo de execução. Neste padrão temos uma classe base que executa ações compartilhadas por todas as subclasses.

Padrões Estruturais – Template Method

Exemplo na Prática: Consulta de Repositórios e Usuários do Git.

- Vamos criar uma aplicação para consultar os repositórios e usuários do Git, utilizando sua API REST.

Padrões Estruturais – Visitor

Orienta a separação de classes de estruturas e de operações, de forma que novos comportamentos possam ser adicionados a objetos sem alterá-los.

- A proposta do Visitor é que criemos classes de estruturas (Ex.: Produtos, Fornecedores e Clientes) e classes de operações de forma isoladas.
- Esse padrão segue esse nome porque os objetos são “visitados” pelas operações.

- **Exemplo:** Temos a exportação de arquivos JSON, XML e CSV de fornecedores e clientes. Essa exportação seria as operações enquanto Fornecedores e Clientes seriam as estruturas.

Padrões Estruturais – Visitor

Elementos desse padrão:

- **Element:** São as classes de estrutura.
- **Visitor:** “Visitantes” que executam as operações.
- **Concrete Element:** Classes que implementam o Element.
- **Concrete Visitor:** Classes que implementam o Visitor.
- **Object Structure:** Conjunto de objetos da classe Concrete Element no qual os visitantes executarão suas operações.

Padrões Estruturais – Visitor

Exemplo na Prática: Cálculo de Aumento de Salário.

- Vamos criar uma aplicação onde realizamos o aumento de salário e identificação da senioridade do funcionário.