MBA USP ESALQ

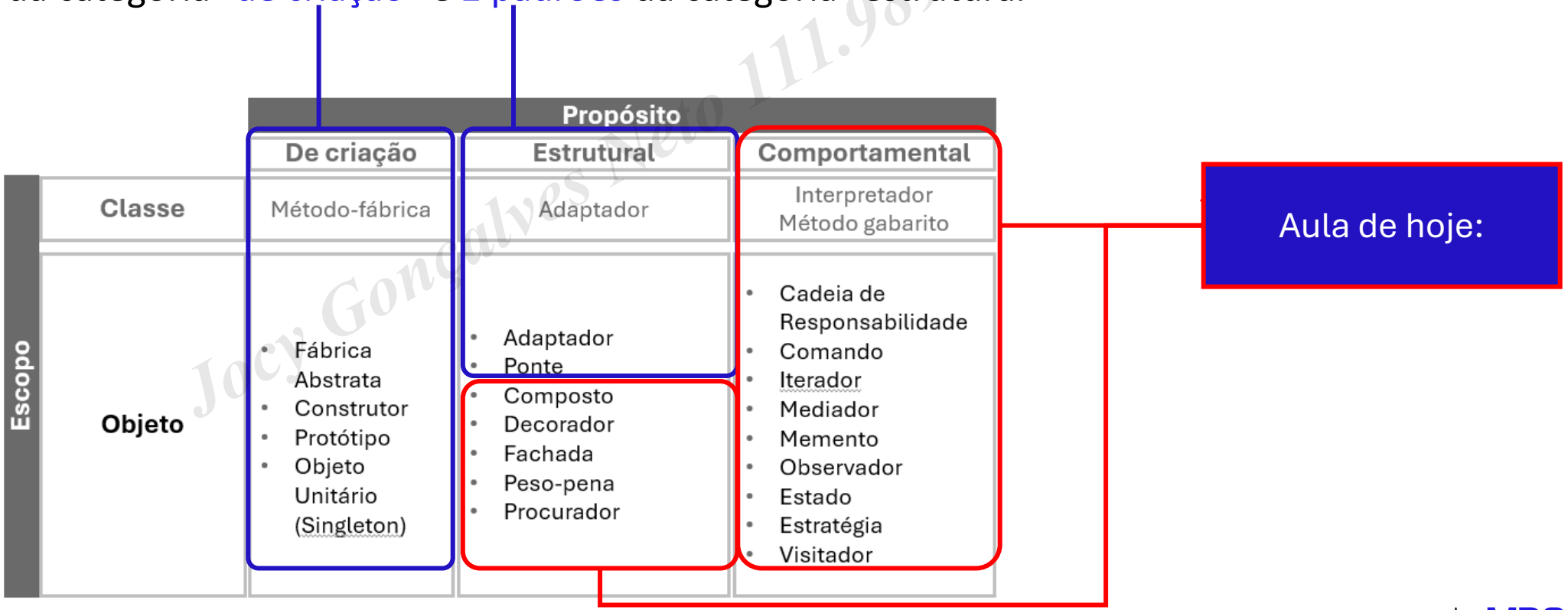
PADRÕES DE PROJETO (DESIGN PATTERNS) II

Profa. Dra. Rosana T. Vaccare Braga

MBA USP ESALQ

Na aula anterior ...

- Reúso de software e seus benefícios
- Padrões e sua origem
- Elementos de um padrão, tipos de padrão
- Padrões de projeto: definições, vantagens, categorias, formatos de escrita
- Padrões GoF da categoria “de criação” e 2 padrões da categoria “estrutural”



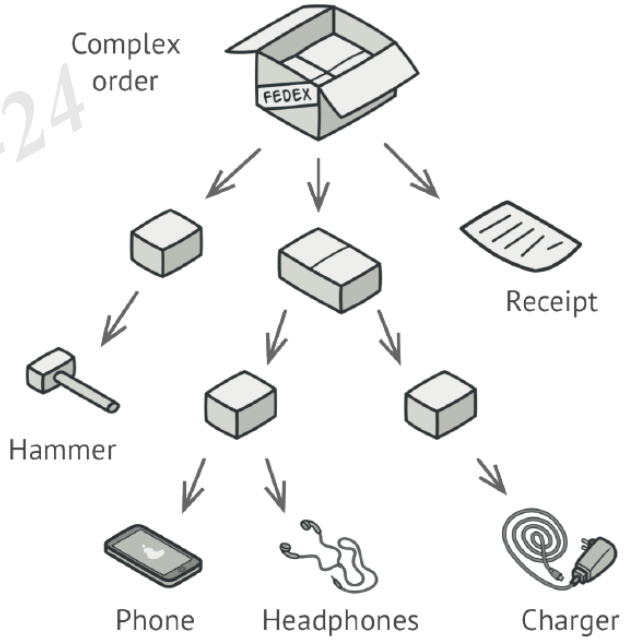
Escopo	Propósito		
	De criação	Estrutural	Comportamental
Classe	Método-fábrica	Adaptador	Interpretador Método gabarito
Objeto	• Fábrica Abstrata • Construtor • Protótipo • Objeto Unitário (Singleton)	• Adaptador • Ponte • Composto • Decorador • Fachada • Peso-pena • Procurador	• Cadeia de Responsabilidade • Comando • Iterador • Mediador • Memento • Observador • Estado • Estratégia • Visitador

PADRÕES da categoria “estrutural”

Preocupam-se com a maneira como ocorrem as composições de classes e objetos em estruturas maiores

Padrão de Projeto Composto (do inglês Composite)

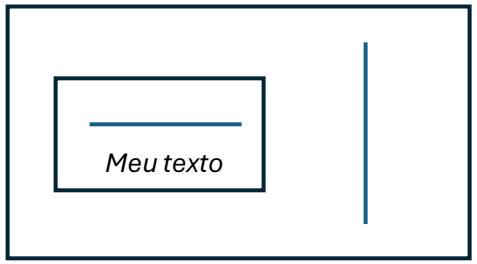
- **Classificação:** Padrão de Criação/Escopo de objetos
- **Intenção**: Compor objetos em estruturas de árvore para representar hierarquias todo-parte. O Composto permite que o cliente trate objetos individuais e composições de objetos uniformemente.
- **Motivação**: Em aplicações gráficas, você pode construir diagramas complexos a partir de componentes simples. Pode agrupar componentes para criar outros componentes maiores, em vários níveis.
  - O problema é que o código que usa essas classes deve tratar de maneira diferente os objetos primitivos e os objetos que contêm outros objetos, o que torna o código mais complexo.

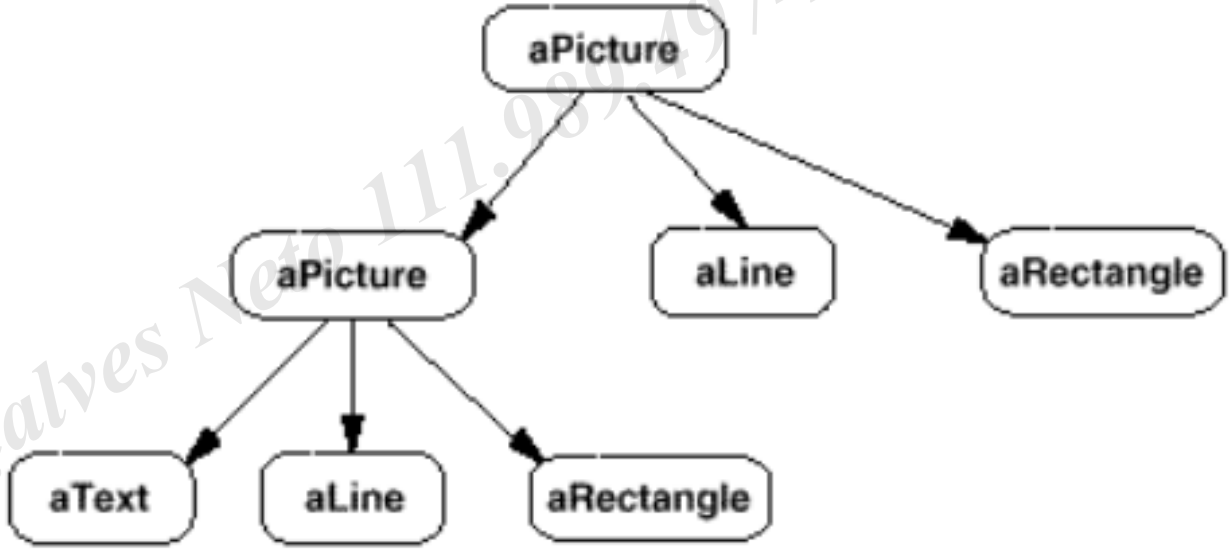


(Shvets, Alexander, 2018)

Enquete 7:
Exemplos de Composto

Outro exemplo do Padrão de Projeto Composto

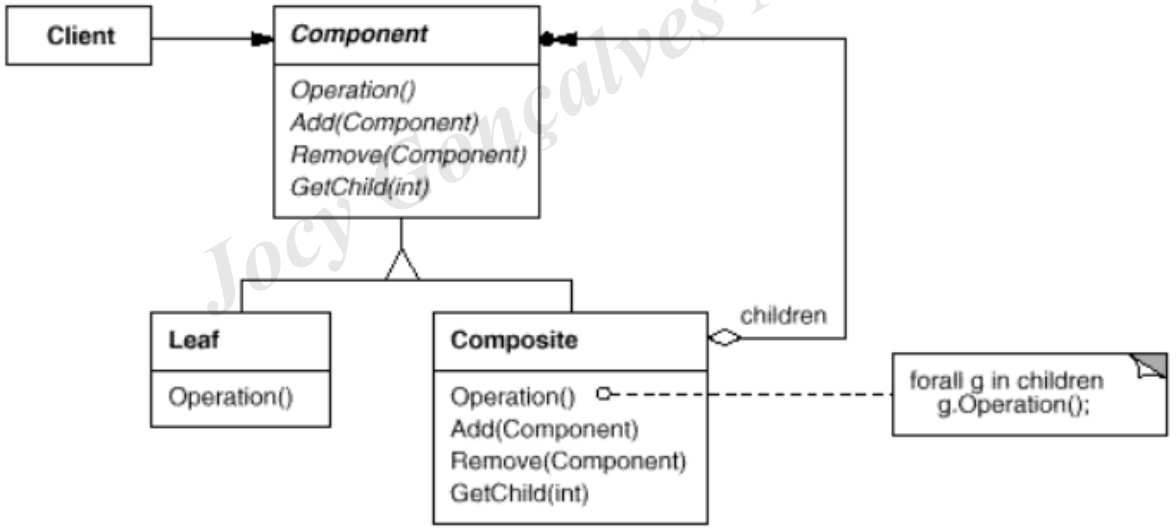


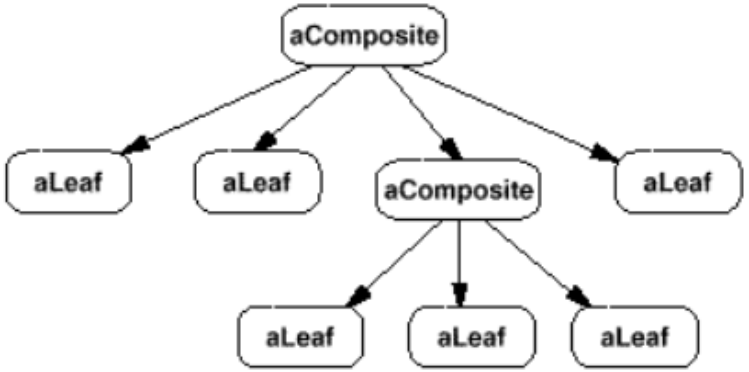


** Imagem capturada no livro versão html publicado em CD (Gamma et al 1995)*

Padrão de Projeto Composto

- **Aplicabilidade**
- Use o padrão Composto quando:
  - você quer representar hierarquias todo-partes.
  - você quer que os clientes sejam capazes de ignorar a diferença entre composições de objetos e objetos individuais. Os clientes tratarão todos os objetos na estrutura composta uniformemente.
- **Estrutura**:





*** Imagens capturadas no livro versão html publicado em CD (Gamma et al 1995)*

Padrão de Projeto Composto

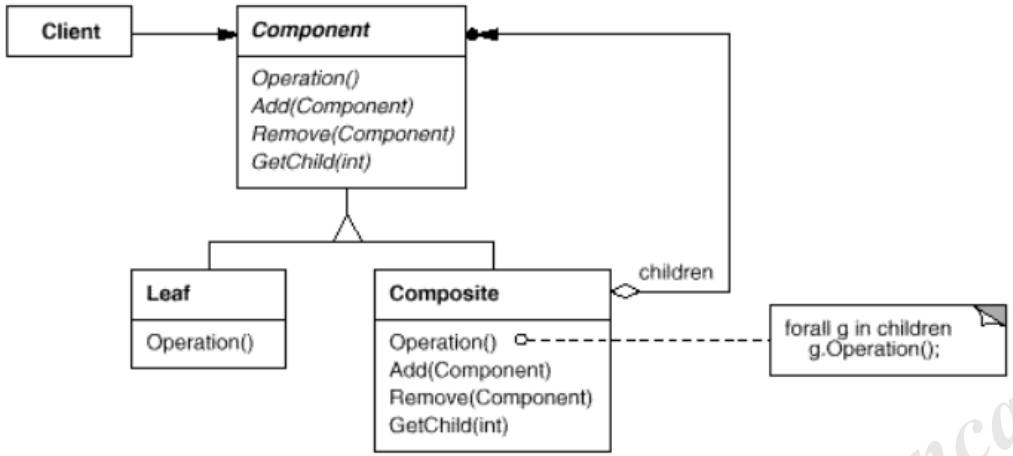
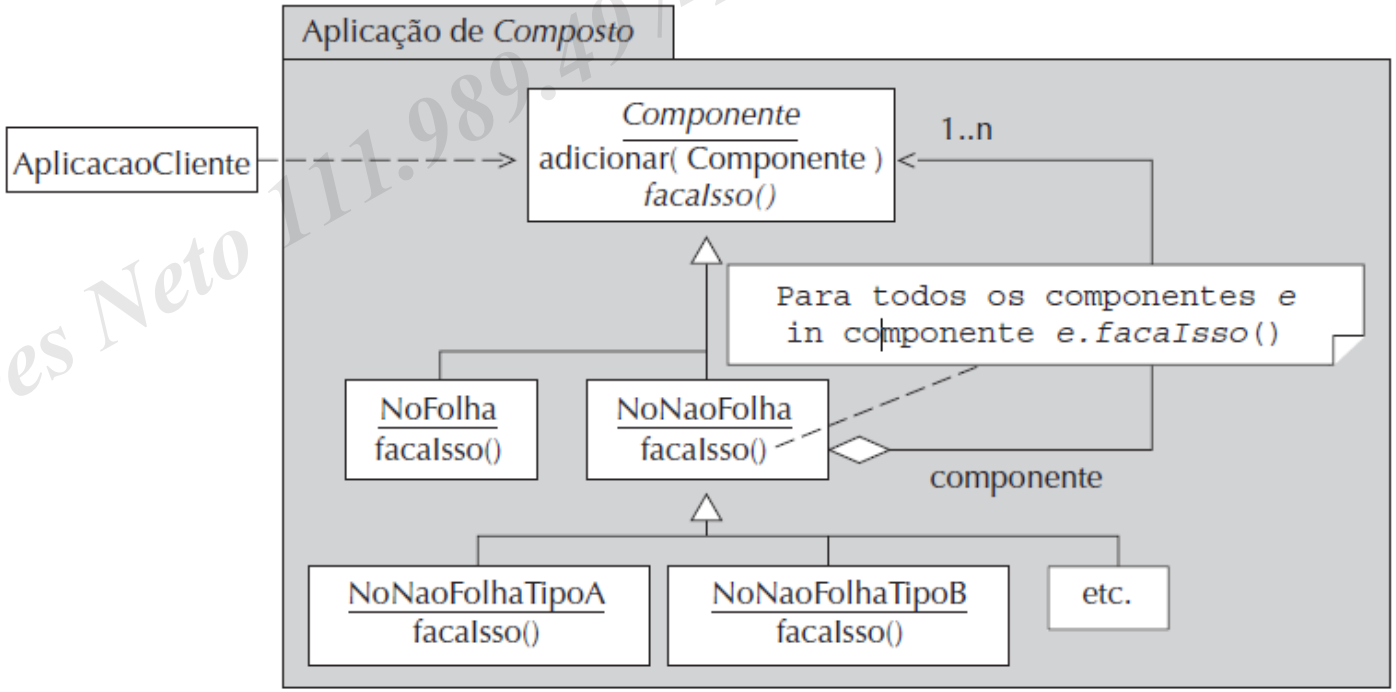


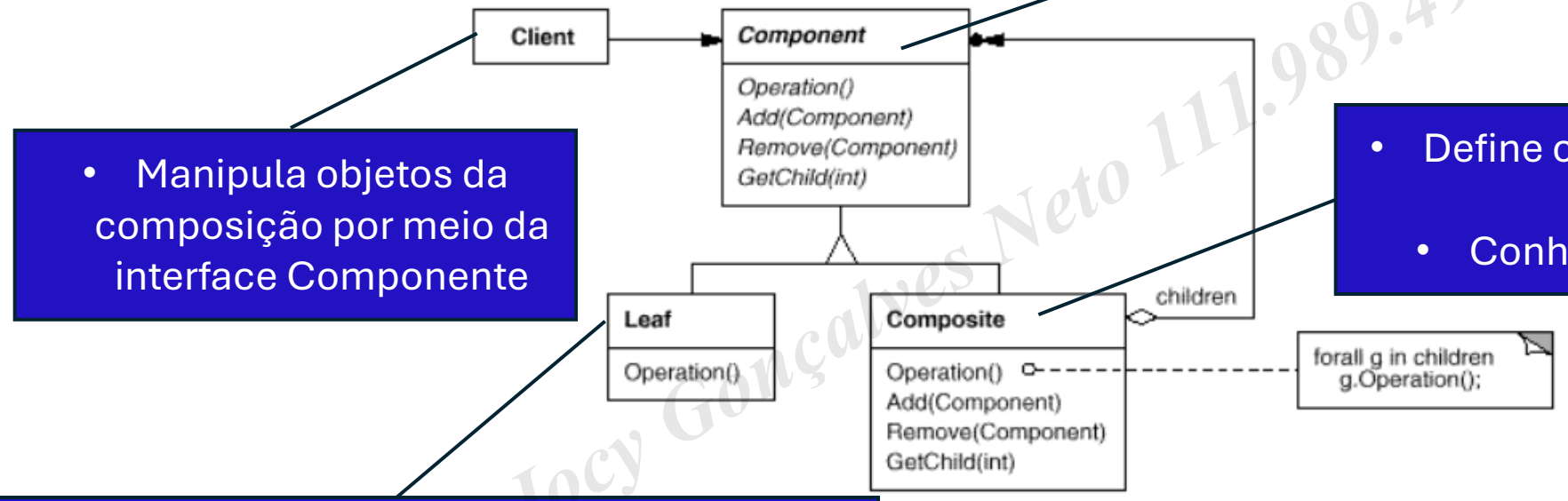
Imagem capturada no livro versão html publicado em CD (Gamma et al 1995)



(Braude, Eric, 2005)

Padrão de Projeto Composto

- **Participantes**:



- Declara interface para objetos da composição e para gerenciar componentes-filho
- Comportamento default fica aqui, se for o caso

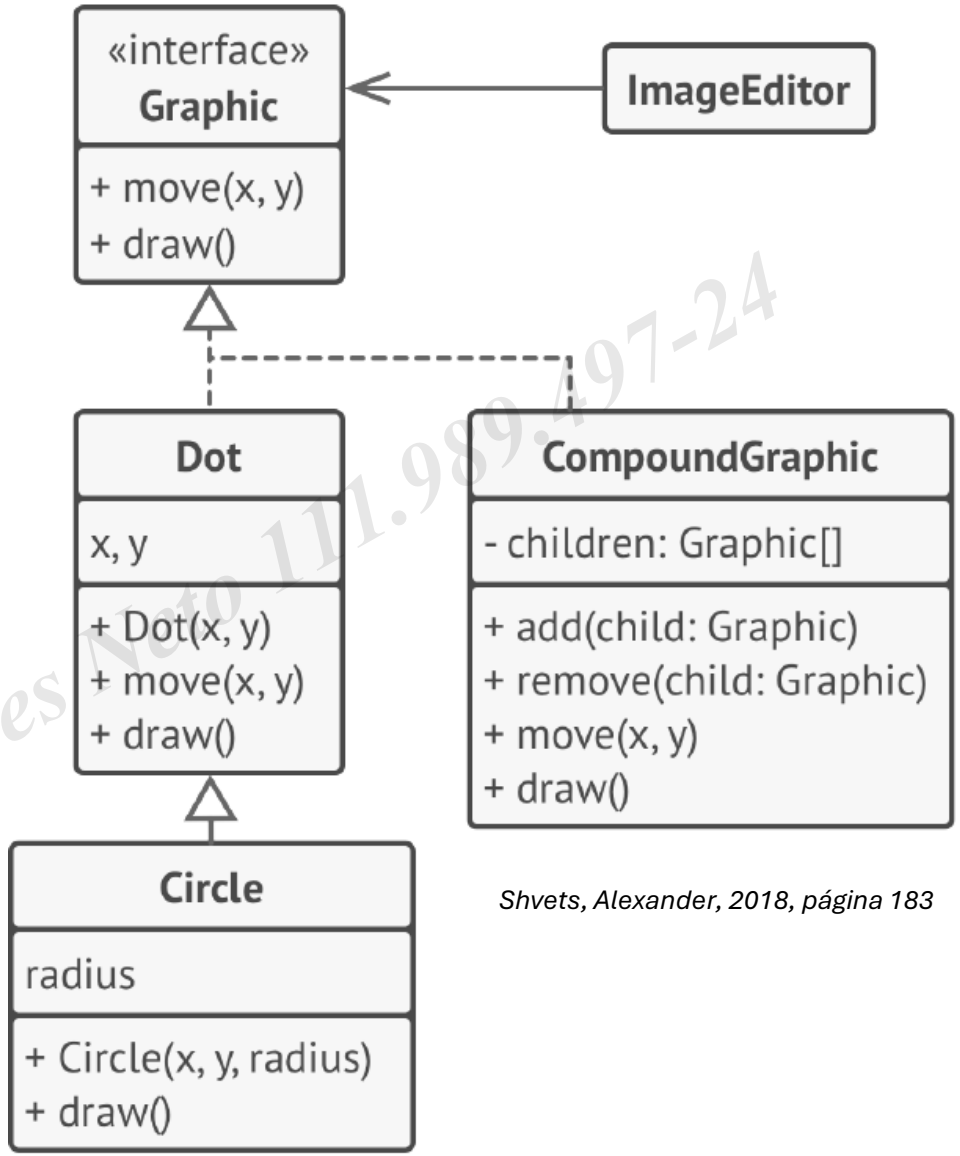
- Manipula objetos da composição por meio da interface Componente

- Define comportamento para objetos que têm filhos
- Conhece os componentes filho

- Representa objetos folha (que não tem filhos)
- Define comportamento para objetos primitivos na composição

*** Imagem capturada no livro versão html publicado em CD (Gamma et al 1995)*

Exemplo de código do Padrão de Projeto Composto



Shvets, Alexander, 2018, página 183

Exemplo de código do Padrão de Projeto Composto

```
// The component interface declares common operations for both
// simple and complex objects of a composition.
interface Graphic is
  method move(x, y)
  method draw()

// The leaf class represents end objects of a composition. A
// leaf object can't have any sub-objects. Usually, it's leaf
// objects that do the actual work, while composite objects only
// delegate to their sub-components.
class Dot implements Graphic is
  field x, y

  constructor Dot(x, y) { ... }

  method move(x, y) is
    this.x += x, this.y += y

  method draw() is
    // Draw a dot at X and Y.

// All component classes can extend other components.
class Circle extends Dot is
  field radius


```


Shvets, Alexander, 2018, página 184

Exemplo de código do Padrão de Projeto Composto

```
27    constructor Circle(x, y, radius) { ... }
28
29    method draw() is
30      // Draw a circle at X and Y with radius R.
31
32  // The composite class represents complex components that may
33  // have children. Composite objects usually delegate the actual
34  // work to their children and then "sum up" the result.
35  class CompoundGraphic implements Graphic is
36    field children: array of Graphic
37
38    // A composite object can add or remove other components
39    // (both simple or complex) to or from its child list.
40    method add(child: Graphic) is
41      // Add a child to the array of children.
42
43    method remove(child: Graphic) is
44      // Remove a child from the array of children.
45
46    method move(x, y) is
47      foreach (child in children) do
48        child.move(x, y)
49
50    // A composite executes its primary logic in a particular
51    // way. It traverses recursively through all its children,
52    // collecting and summing up their results. Since the
53    // composite's children pass these calls to their own
54    // children and so forth, the whole object tree is traversed
55    // as a result.
56    method draw() is
57      // 1. For each child component:
58      //     - Draw the component.
```


Shvets, Alexander, 2018, página 185

Exemplo de código do Padrão de Projeto Composto

```
59      //     - Update the bounding rectangle.
60      // 2. Draw a dashed rectangle using the bounding
61      // coordinates.
62
63
64  // The client code works with all the components via their base
65  // interface. This way the client code can support simple leaf
66  // components as well as complex composites.
67  class ImageEditor is
68    method load() is
69      all = new CompoundGraphic()
70      all.add(new Dot(1, 2))
71      all.add(new Circle(5, 3, 10))
72      // ...
73
74    // Combine selected components into one complex composite
75    // component.
76    method groupSelected(components: array of Graphic) is
77      group = new CompoundGraphic()
78      group.add(components)
79      all.remove(components)
80      all.add(group)
81      // All components will be drawn.
82      all.draw()
```


Shvets, Alexander, 2018, página 186

Consequências de uso do Composto

- Objetos primitivos podem ser compostos em objetos mais complexos, que por sua vez podem ser compostos, e assim por diante recursivamente:
  - onde quer que o código do cliente espere um objeto primitivo, ele também pode receber um objeto composto.
- Torna o cliente simples.
  - Os clientes não precisam saber se estão lidando com uma folha ou um componente composto
  - evita ter que escrever funções no estilo “do case”
- Torna mais fácil adicionar novos tipos de componentes: os clientes não precisam ser alterados para novas classes Componente.
- Pontos negativos:
  - o projeto pode tornar-se muito genérico.
  - fica mais difícil restringir os componentes de um composto. Por exemplo, se for necessário que seu composto tenha apenas certos componentes. Você não pode confiar no sistema de tipos da linguagem de programação para impor essas restrições, mas terá que usar verificações em tempo de execução.

Padrões relacionados ao Composto

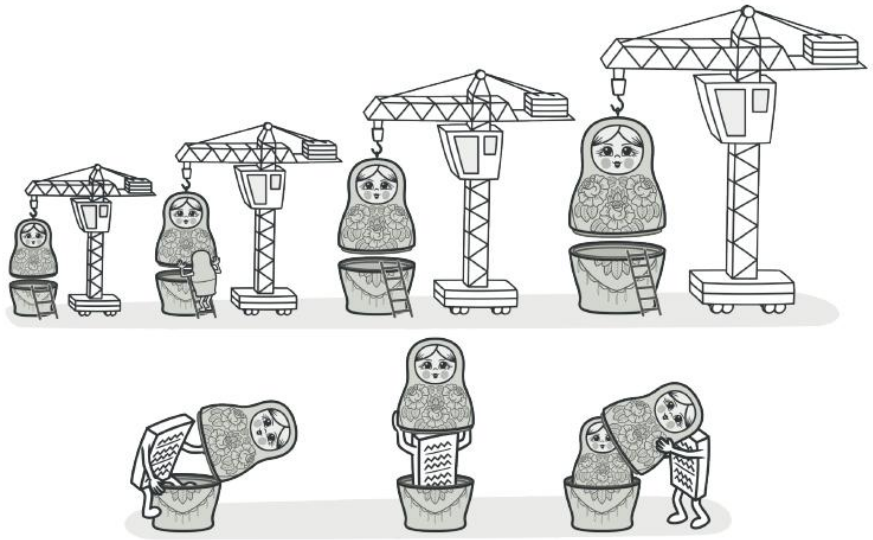
- Chain of Responsibility
- Decorator
- Flyweight
- Iterator
- Visitor

Retomando a enquete 7

- Você consegue desenhar um diagrama de classes aplicando o Padrão Composto em outro exemplo?

Padrão de Projeto Decorador (do inglês Decorator)

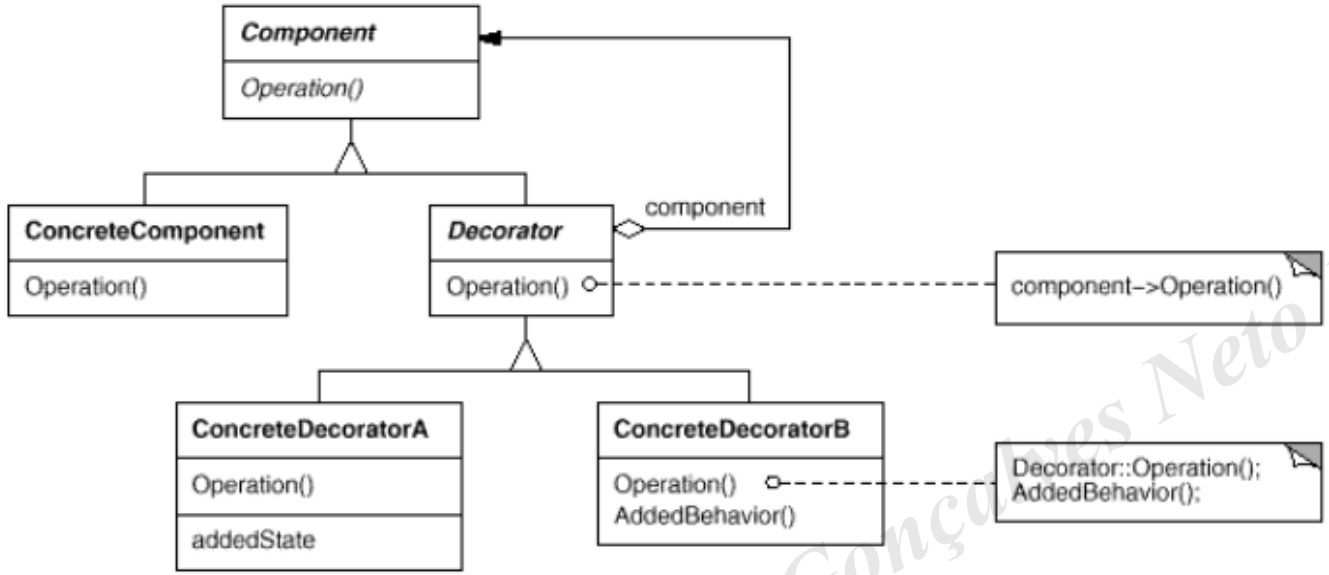
- **Classificação:** Padrão estrutural/Escopo de objetos
- **Intenção**: Atribuir responsabilidades adicionais a um objeto dinamicamente. Decoradores fornecem uma alternativa flexível à herança para estender a funcionalidade.
- **Aplicação**: Use Decorador para:
  - adicionar responsabilidades a objetos individuais de forma dinâmica e transparente, ou seja, sem afetar outros objetos.
  - Permitir que responsabilidades possam ser retiradas.
  - Evitar que um grande número de extensões independentes produza uma explosão de subclasses para dar suporte a cada combinação.



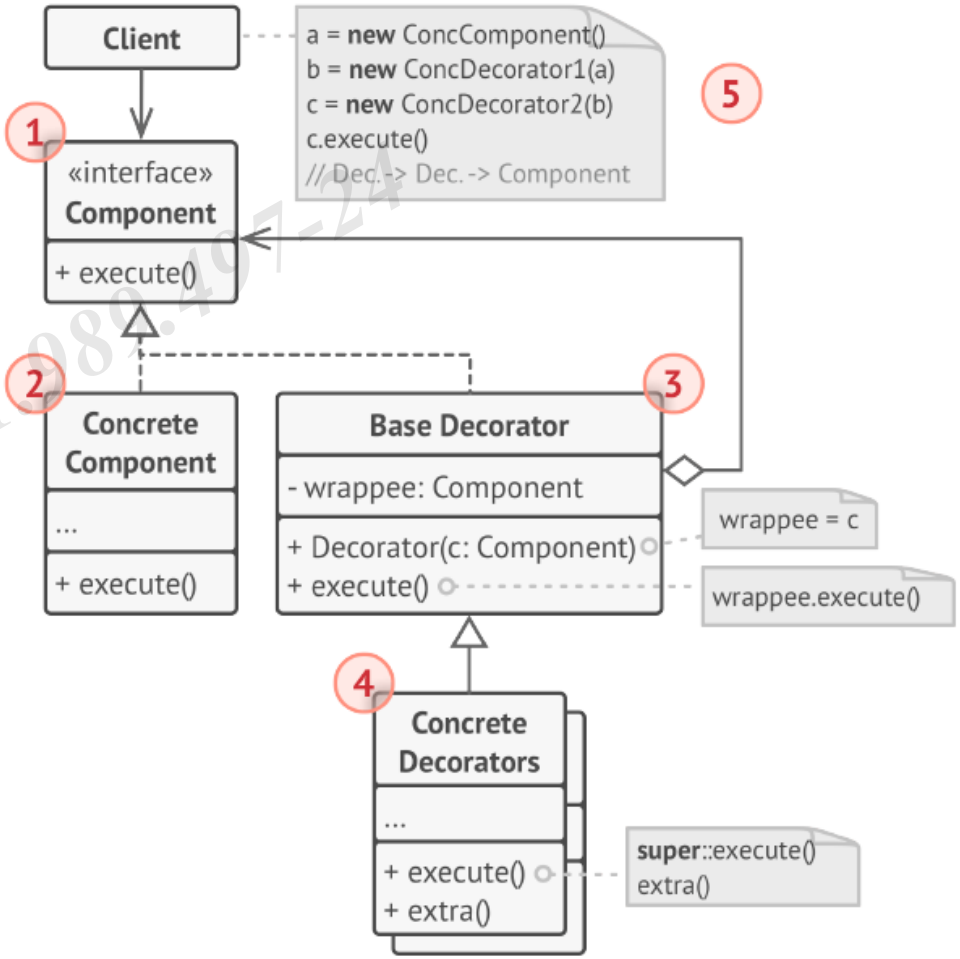
(Shvets, Alexander, 2018)

Padrão de Projeto Decorador

- **Estrutura:**

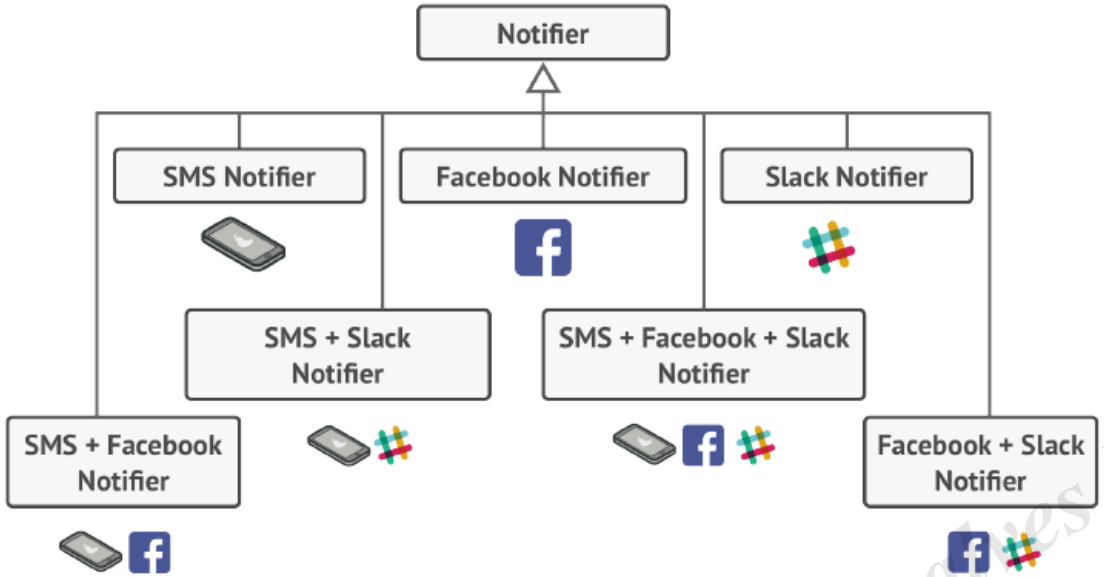


*** Imagem capturada no livro versão html publicado em CD (Gamma et al 1995)*

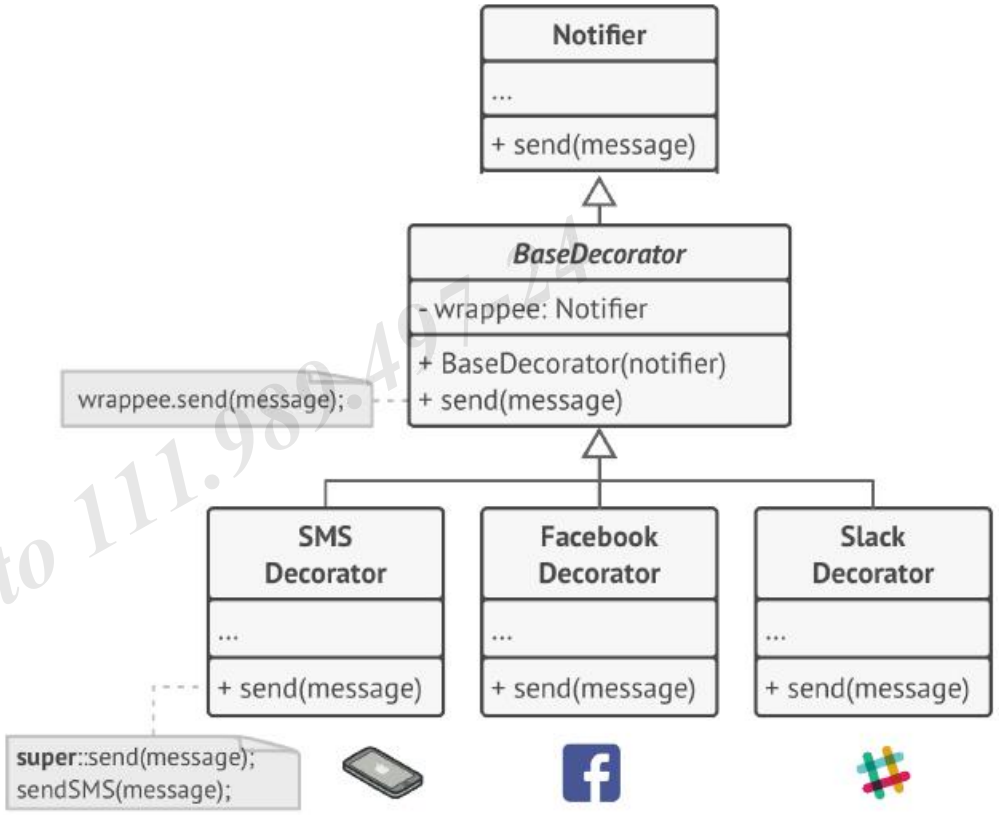


(Shvets, Alexander, 2018)

Padrão de Projeto Decorador



- Sem usar o Decorator

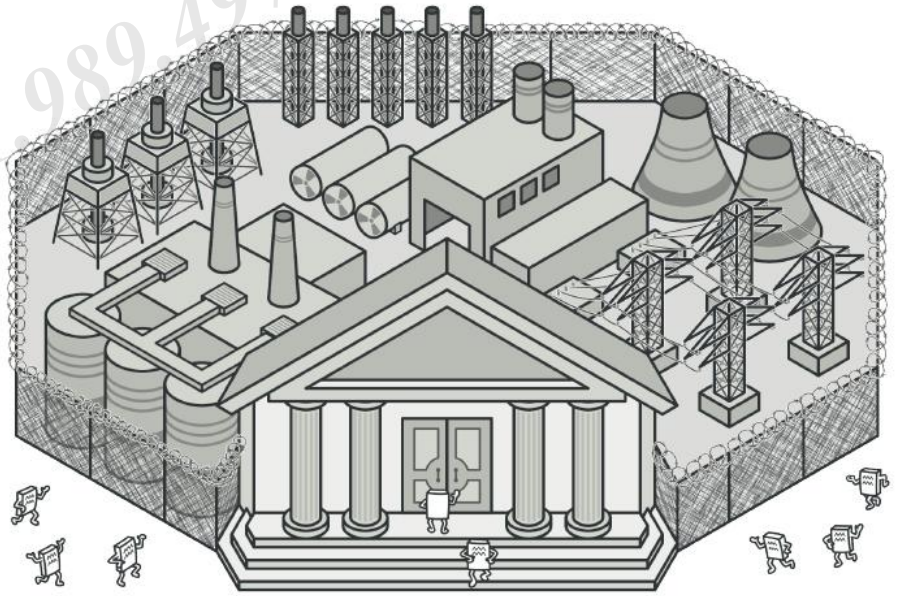


- Usando o Decorator

(Shvets, Alexander, 2018)

Padrão de Projeto Fachada (do inglês Facade)

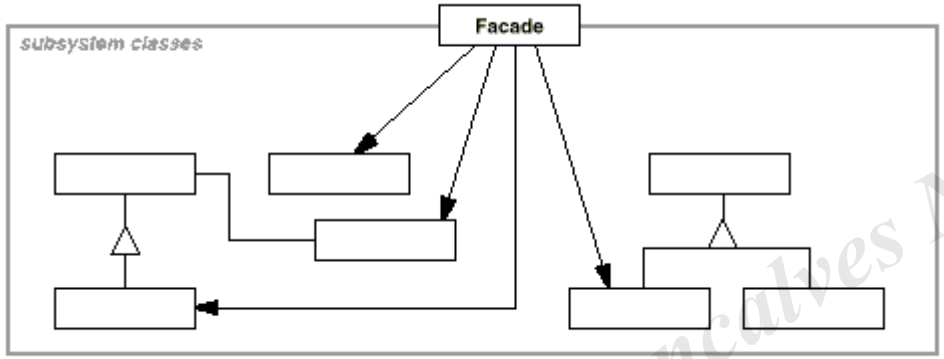
- **Classificação:** Padrão estrutural/Escopo de objetos
- **Intenção**: Fornecer uma interface unificada para um conjunto de interfaces em um subsistema. Fachadas definem uma interface de nível mais alto que torna o subsistema mais fácil de usar.
- **Aplicação**: você precisa integrar seu aplicativo ou sistema com uma biblioteca sofisticada que tem dezenas de recursos, mas você só precisa de uma parte da funcionalidade



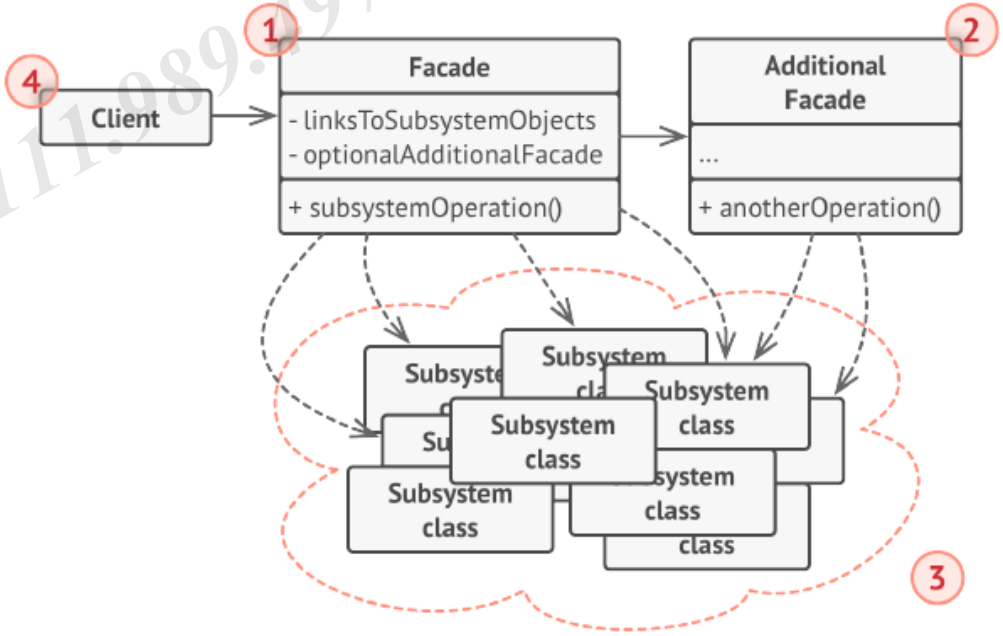
(Shvets, Alexander, 2018)

Padrão de Projeto Fachada

- **Estrutura:**



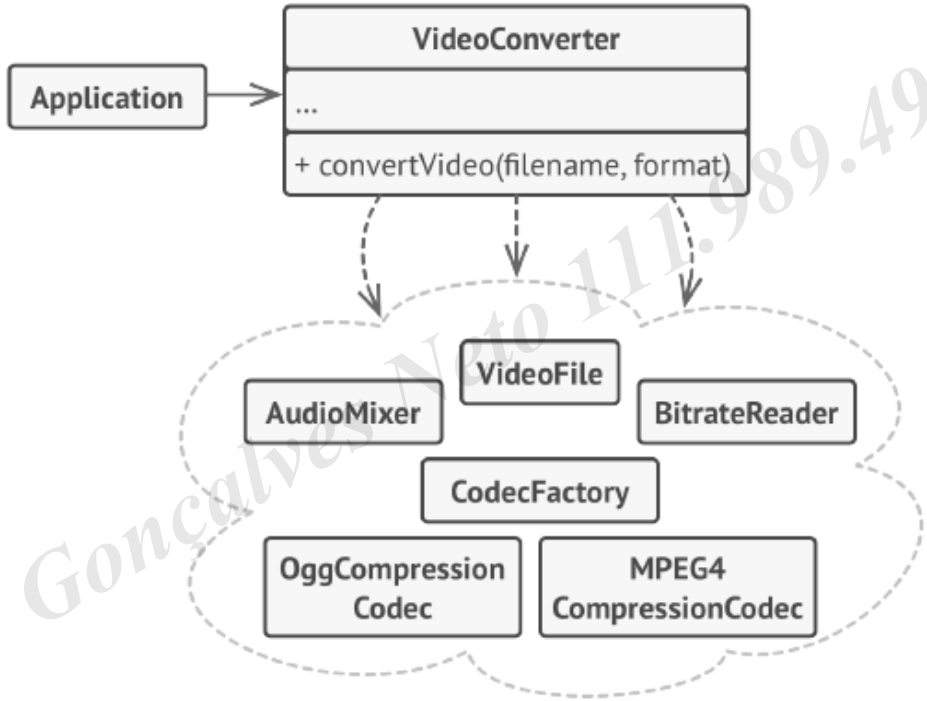
*** Imagem capturada no livro versão html publicado em CD (Gamma et al 1995)*



(Shvets, Alexander, 2018)

Padrão de Projeto Fachada

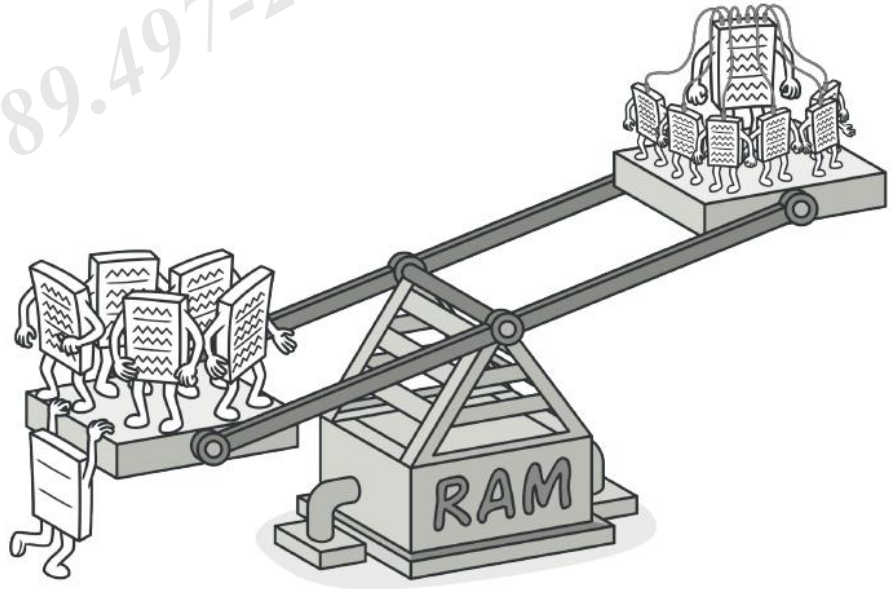
- **Exemplo:**



(Shvets, Alexander, 2018)

Padrão de Projeto Peso-pena (do inglês Flyweight)

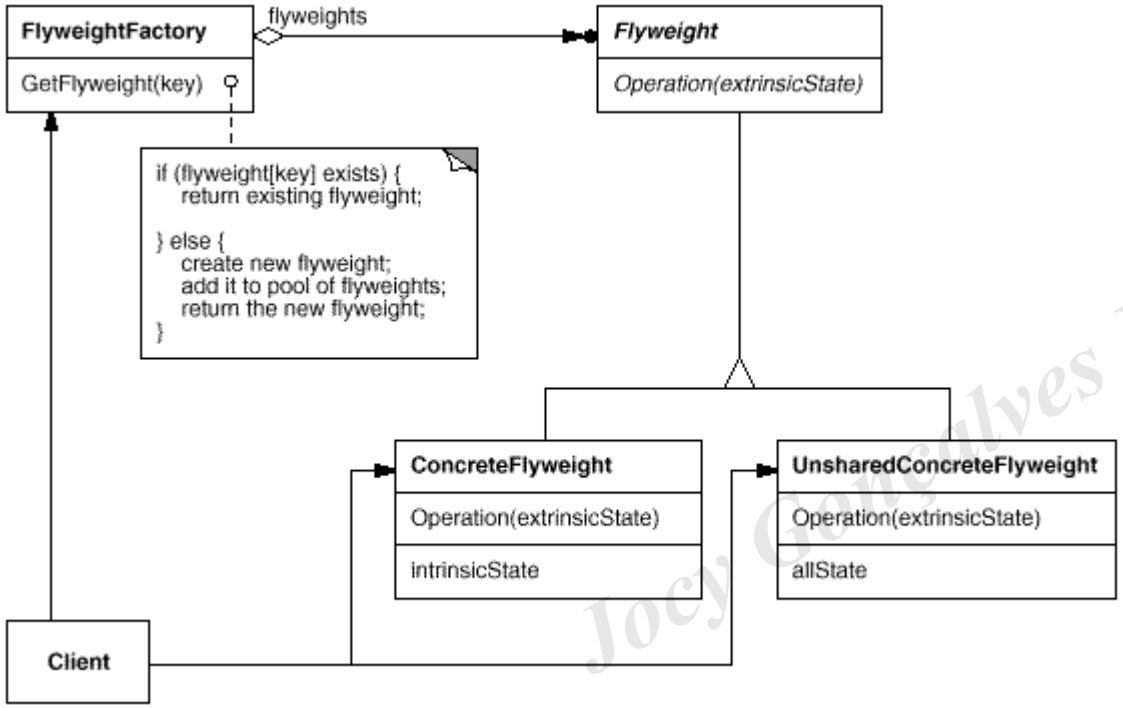
- **Classificação:** Padrão estrutural/Escopo de objetos
- **Intenção**: Usar o compartilhamento para oferecer suporte eficiente a um grande número de objetos de granularidade fina. Esse padrão permite, por exemplo, que você encaixe mais objetos na quantidade disponível de RAM compartilhando partes comuns de estado entre vários objetos em vez de manter todos os dados em cada objeto.
- **Aplicação**: Use o padrão Peso-pena apenas se seu programa precisar suportar um grande número de objetos similares que estão sobrecarregando a RAM disponível. Os objetos devem conter estados duplicados que podem ser extraídos e compartilhados entre vários objetos.



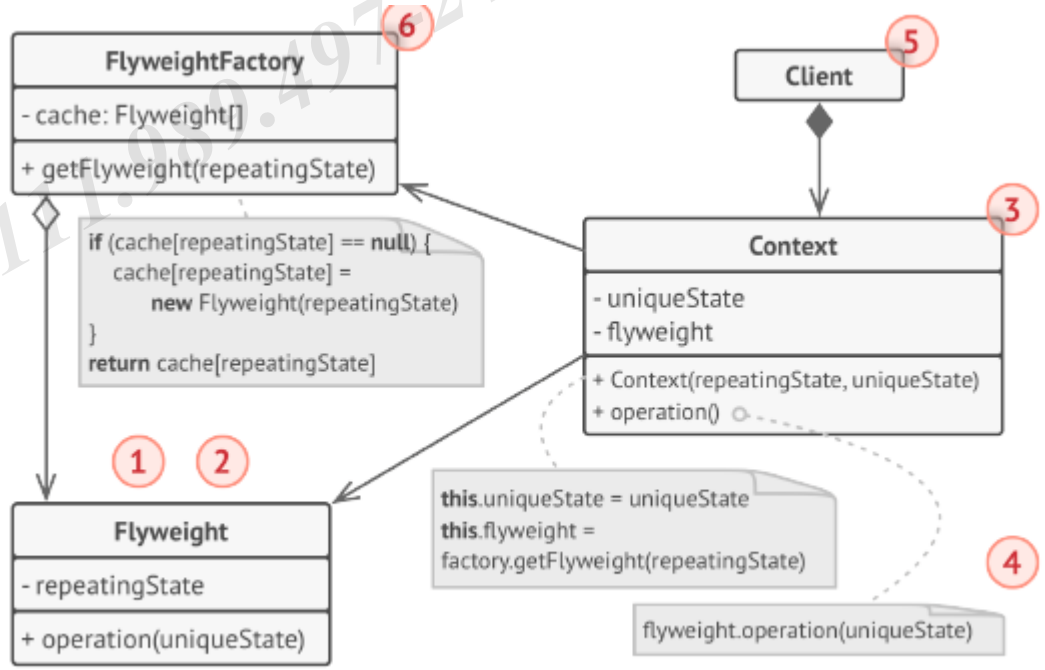
(Shvets, Alexander, 2018)

Padrão de Projeto Peso-pena

- **Estrutura:**



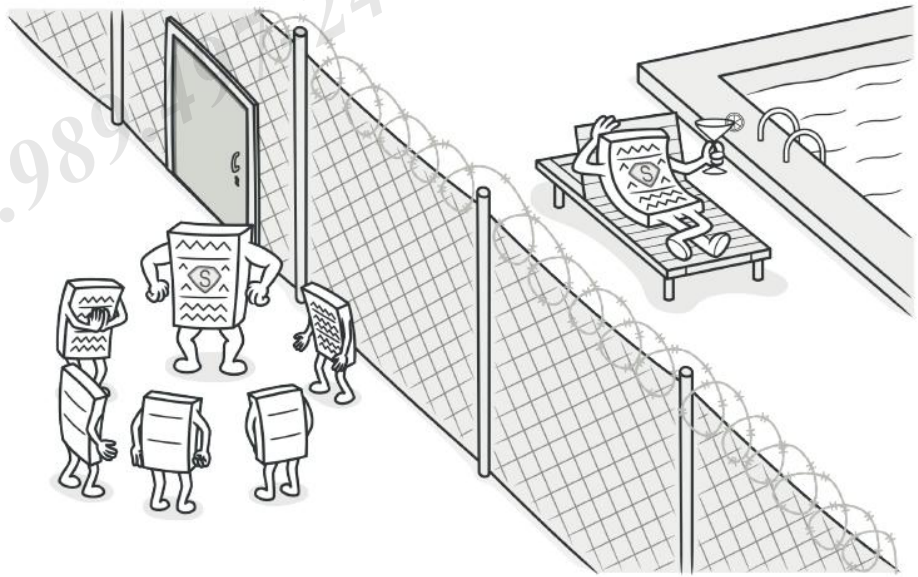
*** Imagem capturada no livro versão html publicado em CD (Gamma et al 1995)*



(Shvets, Alexander, 2018)

Padrão de Projeto Procurador (do inglês Proxy)

- **Classificação:** Padrão estrutural/Escopo de objetos
- **Intenção**: Fornecer um substituto ou espaço reservado para outro objeto para controlar o acesso a ele.
- **Motivação**: Podemos querer controlar o acesso a um objeto, por exemplo, para adiar o custo total de sua criação e inicialização até que realmente precisemos usá-lo. O custo de criação do objeto em geral deve-se ao fato de que muitos atributos precisam serem inicializados, além de dependências com outros objetos que compõem o objeto.



(Shvets, Alexander, 2018)

Padrão de Projeto Proxy

- **Aplicabilidade**
- Usos do padrão Proxy:
  - *Remote proxy*: é necessário fazer referência a um objeto em um espaço de endereçamento diferente,
  - *Virtual proxy*: deseja-se criar objetos sob demanda (adiar a materialização do objeto até que seja realmente necessária), ou "*lazy initialization*"
  - *Protection proxy*: deseja-se manter um controle maior sobre o objeto original (controle de acesso por exemplo),
  - *Smart reference:* deseja-se manter uma referência "esperta" ao objeto (o ponteiro para o objeto realiza algum comportamento adicional sempre que o objeto é acessado).

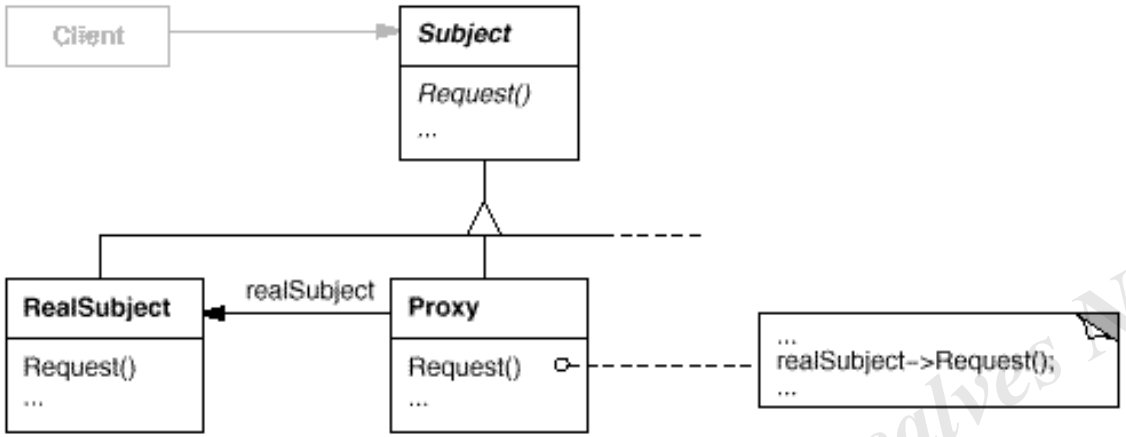
Enquete 8:
Utilização do Proxy

Padrão de Projeto Proxy

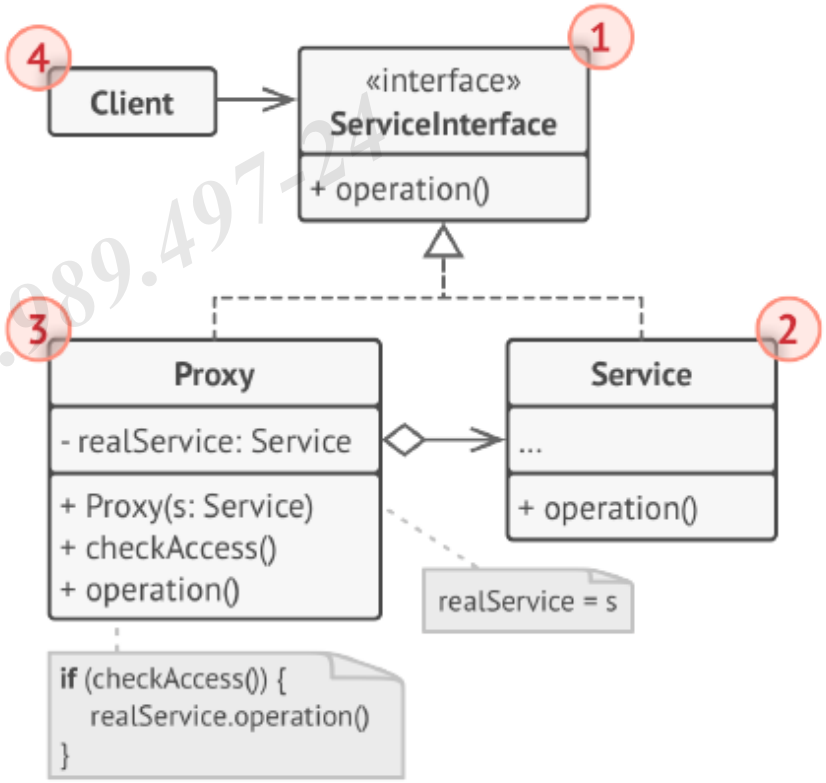
- **Solução**
  - Forneça um substituto, ou procurador, ou ainda um placeholder para o objeto, com a finalidade específica de controlar o acesso ao objeto.
  - O Procurador mantém referência ao Sujeito Real e possui a mesma interface para que possa ser substituído pelo Sujeito Real.
  - Em geral é o Procurador quem cria o sujeito Real, quando necessário.
  - O Procurador delega solicitações ao Sujeito Real somente quando necessário. Isso depende do tipo de Procurador, sendo que vários tipos estão descritos na versão original do padrão Proxy (Gamma et al. 1995).

Padrão de Projeto Proxy

- **Estrutura:**



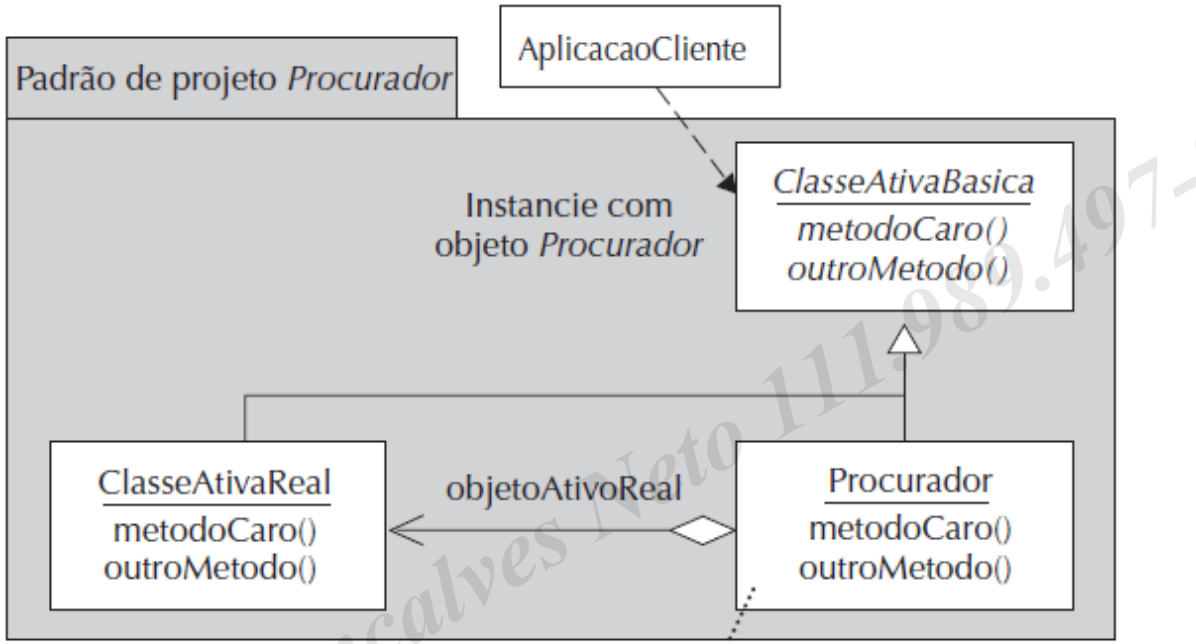
*** Imagem capturada no livro versão html publicado em CD (Gamma et al 1995)*



(Shvets, Alexander, 2018)

Padrão de Projeto Proxy

- **Estrutura:**



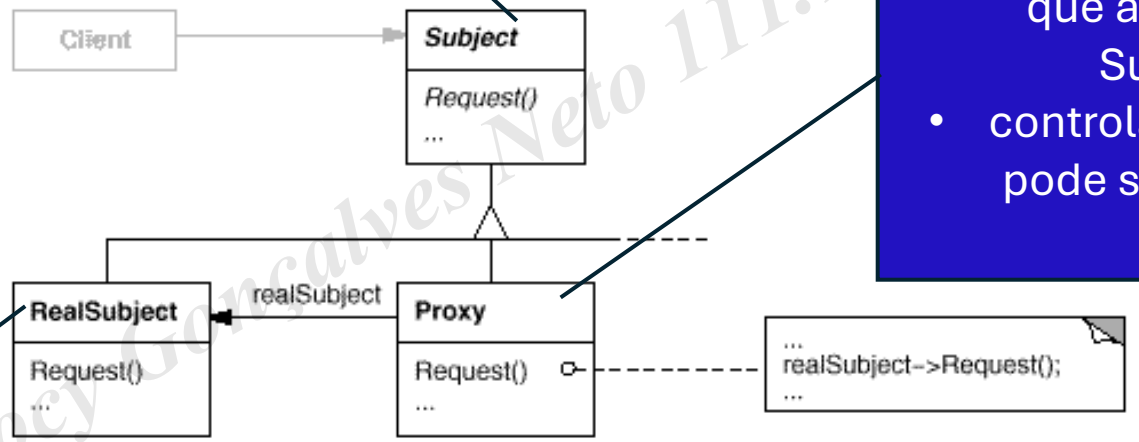

```
...//Uma maneira de verificar se realmente é necessário:
if(objetoAtivoReal==null)   //nunca referenciado
{  objetoAtivoReal = obterObjetoAtivoReal();
   objetoAtivoReal.metodoCaro();
}
else//tenta evitar chamar o metodoCaro() real
```

(Braude, Erick, 2005)

Padrão de Projeto Proxy

- **Participantes**:

Define a interface comum para RealSubject e Proxy, de forma que o Proxy possa ser usado no lugar do RealSubject



- mantém uma referência que permite que o Proxy acesse RealSubject, já que as interfaces RealSubject e Subject são as mesmas.
- controla o acesso ao RealSubject e pode ser responsável por criá-lo e excluí-lo.

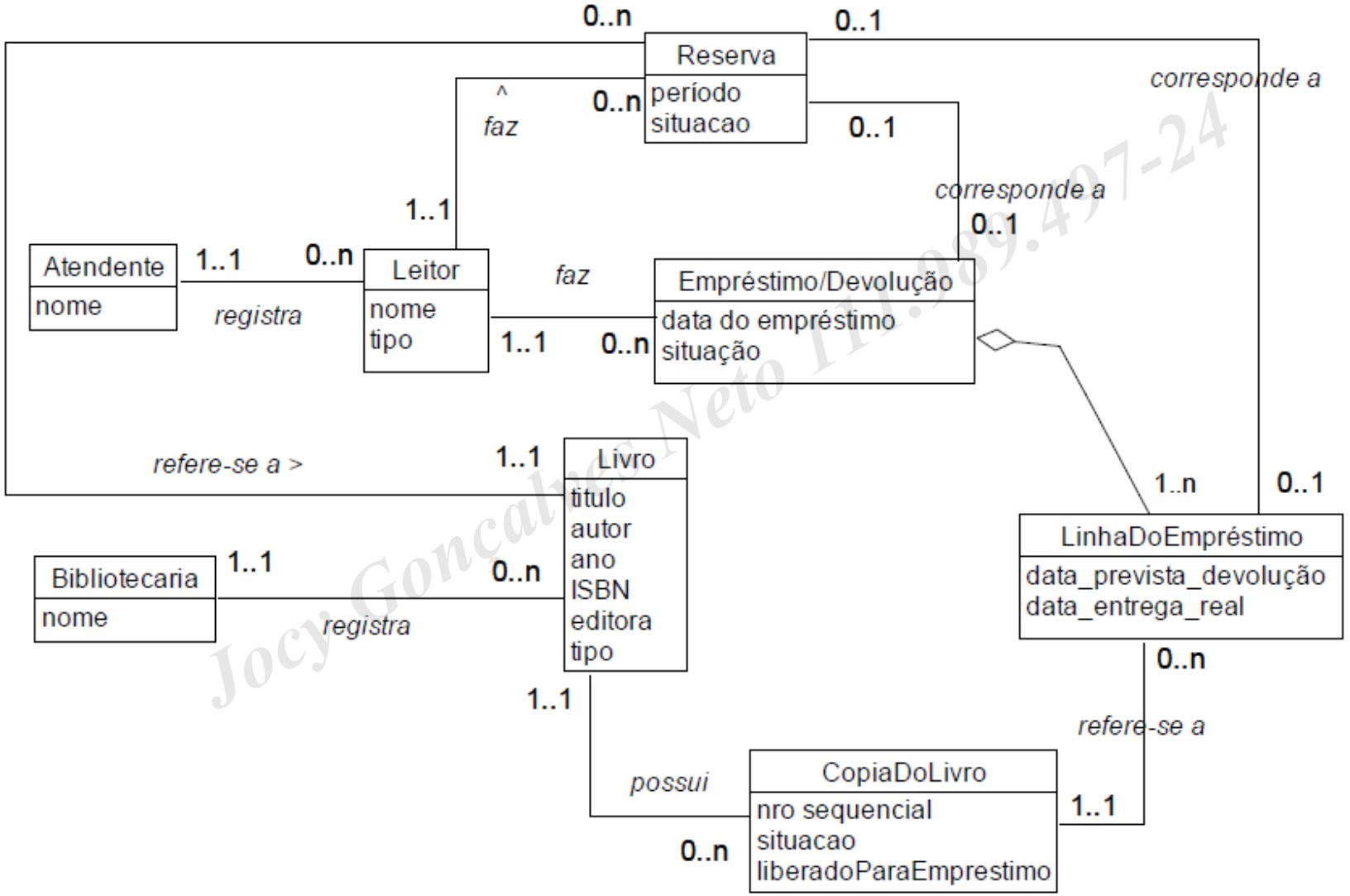
Define o RealSubject que o proxy representa

** *Imagem capturada no livro versão html publicado em CD (Gamma et al 1995)*

Padrão de Projeto Proxy

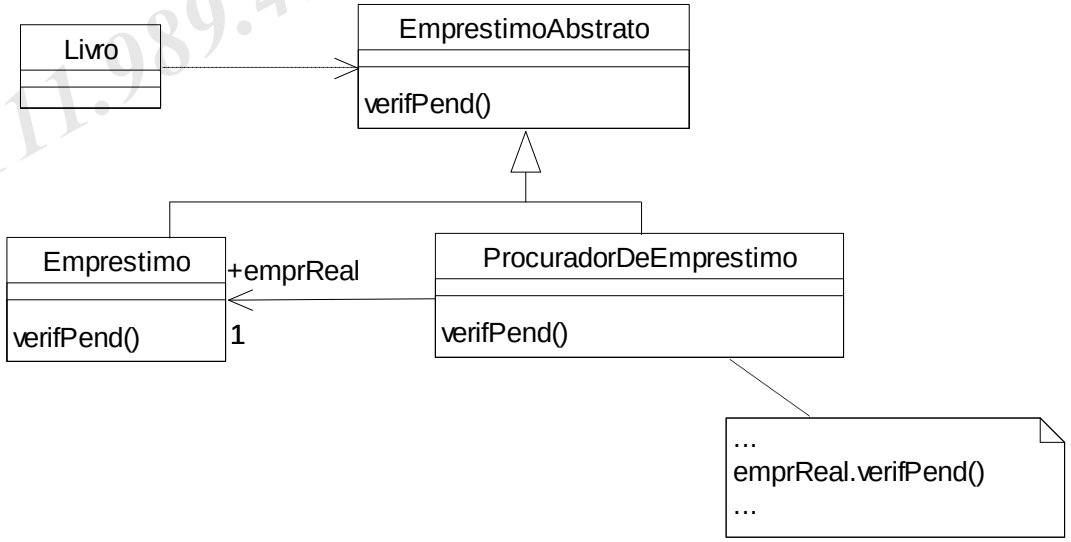
- **Exemplo: materialização e desmaterialização de objetos**
  - Materialização: trazer um registro de uma tabela na forma de um objeto
  - Desmaterialização: persistir um objeto como um registro em uma tabela
- Em um sistema de biblioteca, ao materializar um leitor, podemos no mesmo momento materializar os objetos referentes a todos os empréstimos daquele leitor.
  - No entanto, os objetos empréstimo talvez sejam materializados inutilmente, pois o sistema provavelmente não utilizará informações sobre esses objetos, desperdiçando tempo e espaço.

Padrão de Projeto Proxy

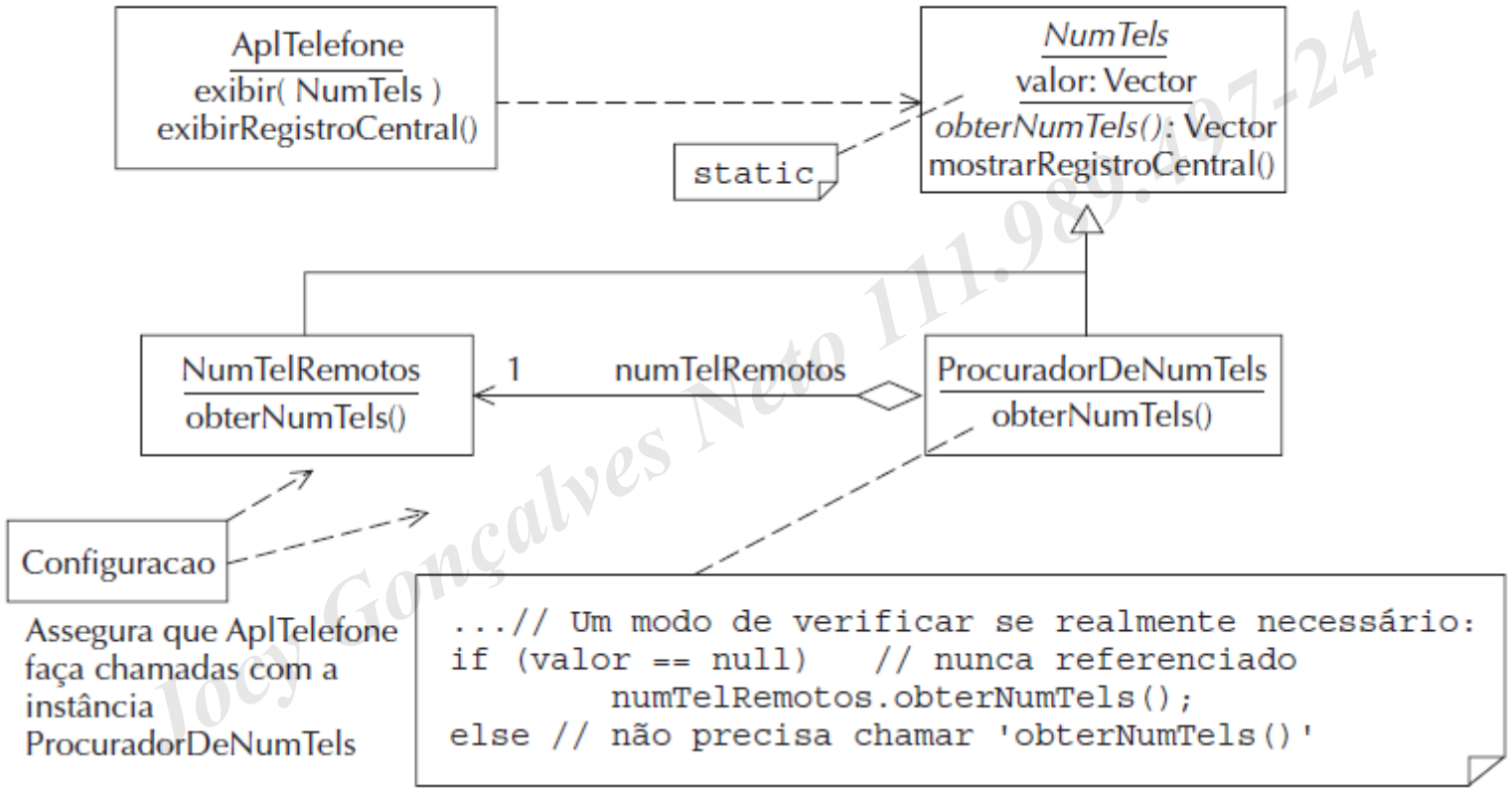


Padrão de Projeto Proxy

- A variante **Procurador Virtual** pode ser usada para solucionar este problema
- Quais objetos associados ao objeto X devem ser recuperados toda vez que um objeto X é recuperado?
- Para os objetos que não precisarão ser recuperados, cria-se um procurador.
  - Exemplo: ao materializar um Livro, deve-se também materializar seus empréstimos associados?
    - Resposta: não



Outro exemplo do Padrão de Projeto Proxy



(Braude, Erick, 2005)

Trechos de Código para o exemplo (Fonte: Braude, 2005)

```
/ ** ============== C L A S S E AplTelefone  ==================
* Aplicação Cliente de 'NumTels'
* Nota de projeto: Essa classe deliberadamente não está ciente se os dados necessários
são remotos ou não.
*/
class AplTelefone
{
  public AplTelefone()   // construtor
  { super();
  }
public static void exibir( NumTels umNumTels )
  { umNumTels.obterNumTels();
  }
public static void exibirRegistroCentral( NumTels umNumTels )
  { umNumTels.mostrarRegistroCentral();
  }
}
```

Continuação do código

```
/ ** =============== C L A S S E    Configuracao ================
* Configura a aplicação-cliente para executar com o procurador
*/
import java.util.*;
import java.io.*;
class Configuracao
{
  public Configuracao()
  { super();
  }
  / ************ M É T O D O obterComandoDoUsuario *********
  * Pós-condição: Foi solicitado ao usuário inserir 'todos', 'central' ou 'sair'
  *
  * Retorna: "t" ou "c" ou "s" se o usuário inserir 'todos', 'central' ou 'sair'
  * respectivamente, caso contrário retorna "t"
  */
  public static String obterComandoDoUsuario()
  {
  String comando = "t"; // default
  Hashtable tabelaDeComandos = new Hashtable(); // chaves para entrada do usuário
  // Entrada de chave pelo usuário para saída correspondente
```

Continuação do código

```
tabelaDeComandos.put( "todos", "t" );
tabelaDeComandos.put( "central", "c");
tabelaDeComandos.put( "sair", "s" );
// Informa ao usuário o que selecionar
System.out.println( "\nPor favor, selecione um comando entre os seguintes:" );
for ( Enumeration enumeration = tabelaDeComandos.keys();
  enumeration.hasMoreElements();)
  { System.out.println( enumeration.nextElement() );
  }
// Obtém o tipo de comando do usuário
try // escolhe a entrada do usuário
  {
    BufferedReader bufReader =
    new BufferedReader( new InputStreamReader( System.in ) );
    comando = bufReader.readLine();
    System.out.println(); // linha em branco para separar a entrada dos resultados
  }
catch( IOException e )
  { System.out.println( e );
  }
```

Continuação do código

```
// Retorna se válido – caso contrário assume o padrão
String comandoARetornar = (String)tabelaDeComandos.get( comando );
if( comandoARetornar == null ) // erro de digitação do usuário
  {
    return "a";
  }
else
  { return comandoARetornar;
  }
}
/ ********************** M É T O D O main ***********
* Pós-condições:
* (1) O prompt ao usuário se repete como em 'obterComandoDoUsuario()' até que "s" seja inserido
* (2) Sempre que o usuário responde com "t", todos os registros de telefone são exibidos no console
* (3) Sempre que o usuário responder com "c", os registros de telefone "centrais" são
* exibidos no console como especificado em 'exibirRegistroCentral()'.
*
* Notas de projeto: Utilização do padrão de projeto Procurador.
* Os números telefônicos são recuperados da Internet somente quando necessário
* (cuidado por 'aplTelefone.exibir()' e 'aplTelefone.exibirRegistroCentral()').
*/
```

Continuação do código

```
public static void main( String[] args )
{
AplTelefone aplTelefone = new AplTelefone();
ProcuradorDeNumTels procuradorDeNumTels = new ProcuradorDeNumTels(); // chamadas para
//'NumTels' serão feitas com isso
String requisicaoDoUsuario = "Não atribuído ainda";
while(!"s".equals( requisicaoDoUsuario ) ) // não quer fechar
  {
  requisicaoDoUsuario = obterComandoDoUsuario(); // obtém o próximo comando
  if( "t".equals( requisicaoDoUsuario ) ) // mostra tudo
    { aplTelefone.exibir( procuradorDeNumTels );
    }
  if( "c".equals( requisicaoDoUsuario ) ) // mostra o(s) registro(s) central(is)
    { aplTelefone.exibirRegistroCentral( procuradorDeNumTels );
    }
    // Espaço aqui para outros comandos....
  }
} // fim de main()
```

Continuação do código

```
/ ** ======================== C L A S S E    NumTelRemotos ====================
* Acesso à URL que contém as informações de telefone
*/
import java.net.*;
import java.io.*;
import java.util.*;
class NumTelRemotos extends NumTels
{
public static final String URL_CONTENDO_REGS_TELS =
"http://metcs.bu.edu/~ebraude/SoftwareDesignBook/telephones.txt"; // localização dos dados
public NumTelRemotos()
{ super();
}
/ ********************** M É T O D O obterNumTels ***************************
* Pós-condição: Os objetos 'String' em 'Telnums.valor' consistem em registros
* obtidos na URL 'URL_CONTENDO_REGS_TELS', se uma conexão com
* 'URL_CONTENDO_REGS_TELS' foi estabelecida.
*
* Nota de projeto: Dados recuperados da Internet
*/
```

Continuação do código

```
public void obterNumTels()
{
  System.out.println( "================= Recuperando da Internet ==================" );
  BufferedReader leitorDeURL = null;
  try
  {
    NumTels.valor = new Vector(); // de agora em diante não 'null'
    // Prepara-se para ler da URL
    URL url = new URL( URL_CONTENDO_REGS_TELS );
    URLConnection urlConnection = url.openConnection();
    leitorDeURL = new BufferedReader( new InputStreamReader( urlConnection.getInputStream() ) );
    // Lê os registros de telefone
    String linhaDeURL = "";
    while( linhaDeURL!= null ) // lê até que não haja mais linhas
      {
        linhaDeURL = leitorDeURL.readLine();
        if( linhaDeURL!= null ) // não mantém a leitura de nulo
          {
            NumTels.valor.addElement( linhaDeURL ); // mantém o registro
          }
      }
    leitorDeURL.close();
  }
```

Continuação do código

```
  catch( IOException e )
    { System.out.println( e );
    }
  } // fim de obterNumTels()
}

/ ** ================ C L A S S E    NumTels ===========
* Classe básica para as informações sobre telefone */
import java.util.*;
abstract class NumTels
{
  public static Vector valor = null; // registros mantidos aqui; 'null' indica ainda não recuperado
  public NumTels()
  { super();
  }
  / *************** M É T O D O exibirNumTels ***************
  public void exibirNumTels()
  {
  for( int i = 0; i < NumTels.valor.size(); ++i )
    { System.out.println( (String)NumTels.valor.elementAt( i ) );
    }
  }
```

Continuação do código

```
/ ********************** M É T O D O obterNumTels **********************
*/
public abstract void obterNumTels();
/ ******************** M É T O D O mostrarRegistroCentral ****************
public void mostrarRegistroCentral()
{
  if( valor == null )
  { System.out.println( "mostrarRegistroCentral() chamado com 'valor' nulo" );
  }
else
  {
  int valorDoMeio = (int)( valor.size() / 2 ); // arredonda para cima se 'valor.size()' for ímpar
  if( 2*valorDoMeio!= valor.size() ) // comprimento de 'valor' é ímpar
    { System.out.println( valor.elementAt( valorDoMeio ) );
    }
  else // comprimento é par
    { // Mostra o par em ambos os lados do meio
      System.out.println( valor.elementAt( valorDoMeio - 1) );
      System.out.println( valor.elementAt( valorDoMeio ) );
    }
  }
}
}
```

Continuação do código

```
** ====================== C L A S S E ProcuradorDeNumTels ======================
* Trata do trabalho realmente demorado de recuperação através da Internet
*/
class ProcuradorDeNumTels extends NumTels
{
  NumTelRemotos numTelRemotos = new NumTelRemotos(); // instancia somente se necessário
  public ProcuradorDeNumTels()
  { super();
  }
  / *************** M É T O D O obterNumTels ************************
  * Pós-condição:
  * (1) 'valor' tem os registros de telefone como especificado por 'obterNumTels()' de 'NumTelRemotos'
  * (2) Se os valores já foram recuperados, é exibida uma mensagem no console
  * declarando que o acesso à Internet não é necessário
  * (3) Os registros de telefone foram exibidos no console como ocorre com as
  * pós-condições em 'exibirNumTels()' de 'NumTels'
  *
  * Nota de Projeto: Recupera os registros da Internet somente se eles ainda não foram
  * recuperados.
  */
```

Continuação do código

```
public void obterNumTels()
{
if( valor == null ) // verifica se a recuperação da Internet é necessária
  {
    numTelRemotos = new NumTelRemotos();
    numTelRemotos.obterNumTels();
  }
else // obtém os números de 'NumTels.valor'
  { System.out.println( "======== Não é necessário recuperar da Internet ========" );
}
super.exibirNumTels();
}
/ ********************** M É T O D O mostrarRegistroCentral **********************
* Pós-condições:
* (1) -- como ocorre com 'mostrarRegistroCentral()' em 'NumTels'
* (2) 'NumTels.valor' contém os registros de telefone no site da Internet especificado
* em 'NumTelRemotos'
* (3) Se nenhuma recuperação da Internet for requerida, uma mensagem para esse efeito é
* exibida no console.
*/
```

Continuação do código

```
public void mostrarRegistroCentral()
{
  if( valor!= null )
  {
    System.out.println( "=== Não é necessário recuperar da Internet ===" );
    super.mostrarRegistroCentral();
  }
else // números ainda não recuperados durante essa execução
  {
    numTelRemotos.obterNumTels();
    super.mostrarRegistroCentral();
  }
}
} // fim da classe 'ProcuradorDeNumTels'
```

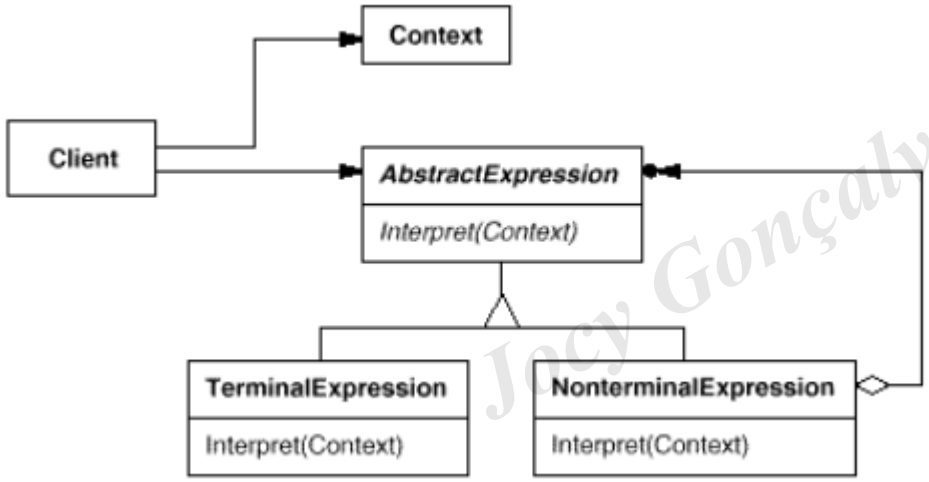
Escopo	Propósito: De criação	Propósito: Estrutural	Propósito: Comportamental
Classe	Método-fábrica	Adaptador	Interpretador Método gabarito
Objeto	• Fábrica Abstrata • Construtor • Protótipo • Objeto Unitário (Singleton)	• Adaptador • Ponte • Composto • Decorador • Fachada • Peso-pena • Procurador	• Cadeia de Responsabilidade • Comando • Iterador • Mediador • Memento • Observador • Estado • Estratégia • Visitador

PADRÕES da categoria “comportamental”

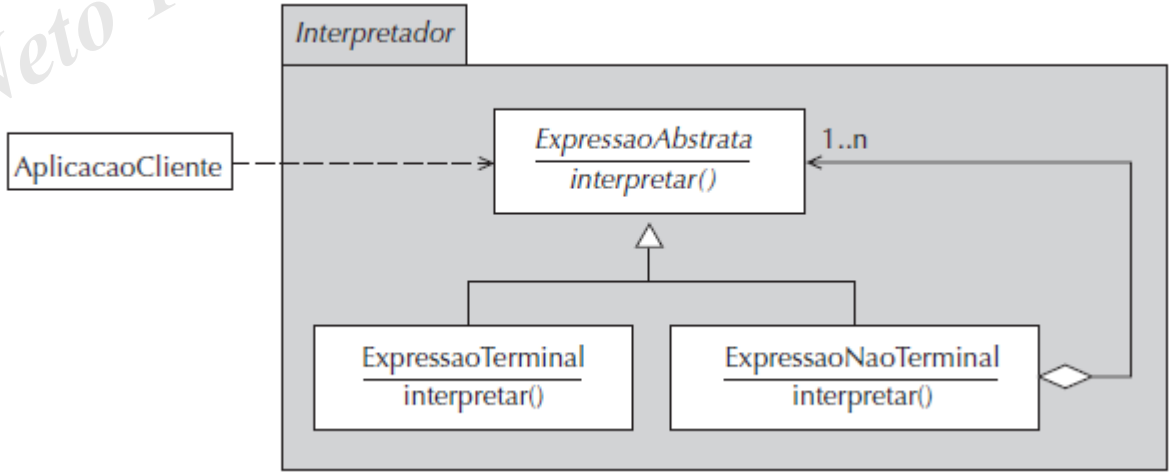
Responsáveis por caracterizar a forma com que as classes e objetos interagem entre si e distribuem as responsabilidades

Padrão de Projeto Interpretador (do inglês Interpreter)

- **Classificação:** Padrão comportamental/Escopo de classes
- **Intenção**: Dada uma linguagem, definir uma representação para sua gramática junto com um interpretador que usa a representação para interpretar sentenças na linguagem.
- **Estrutura**:



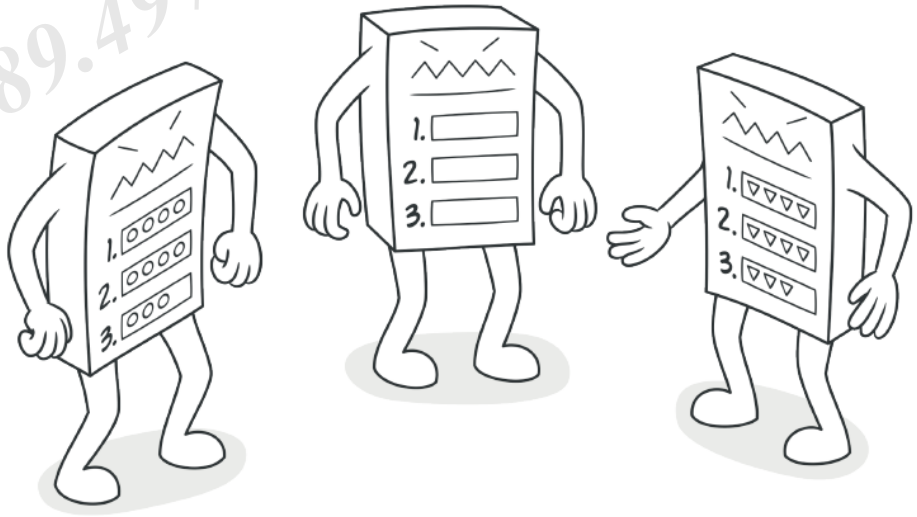
*** Imagem capturada no livro versão html publicado em CD (Gamma et al 1995)*



(Braude, Erick, 2005)

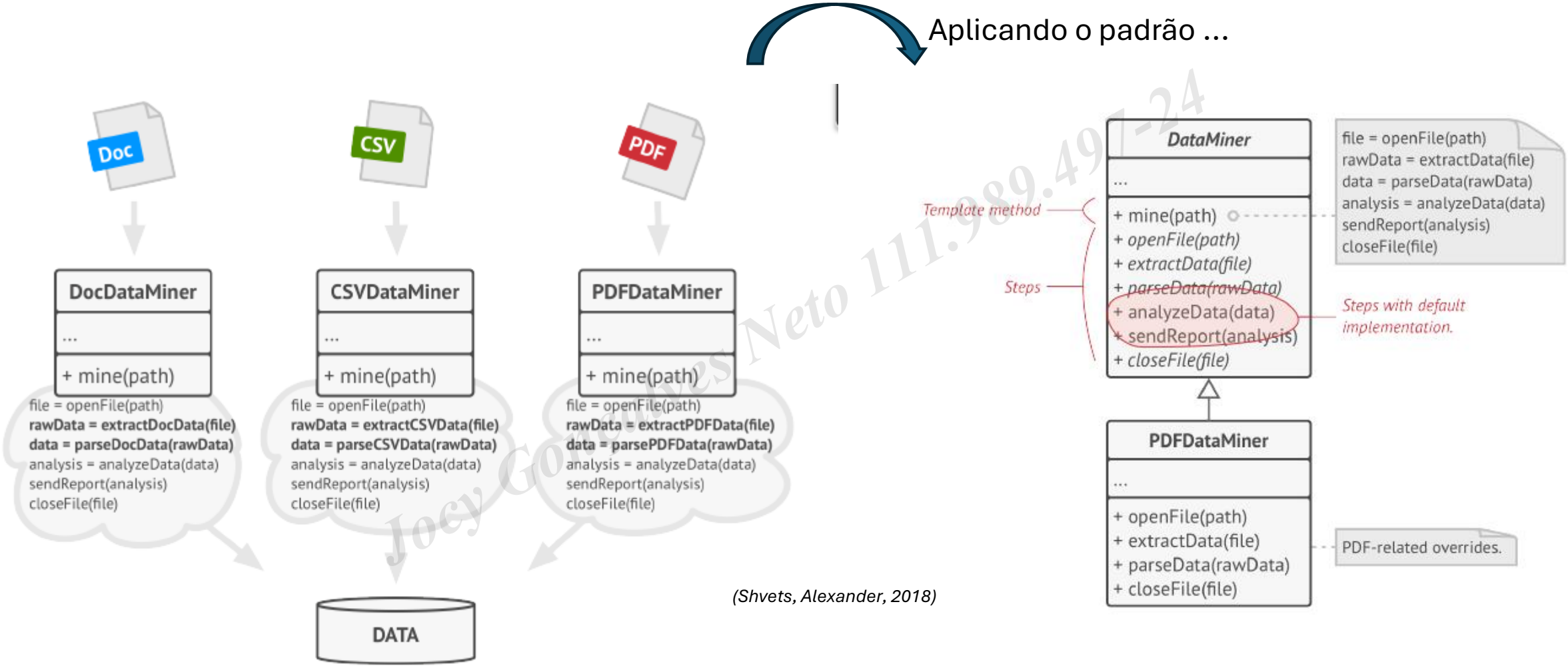
Padrão de Projeto Método-gabarito (do inglês Template Method)

- **Classificação:** Padrão comportamental/Escopo de classes
- **Intenção**: Definir o esqueleto de um algoritmo em uma certa operação, adiando alguns passos para subclasses. O Método-gabarito permite que subclasses redefinam certos passos de um algoritmo sem alterar a estrutura do algoritmo.
  - A ideia do Método-gabarito é definir um método gabarito em uma superclasse, que contenha o esqueleto do algoritmo, com suas partes variáveis e invariáveis. Esse método invoca outros métodos, alguns dos quais são operações que podem ser definidas em uma subclasse.
  - Assim, as subclasses podem redefinir os métodos que variam, acrescentando comportamento específico dos pontos variáveis.



(Shvets, Alexander, 2018)

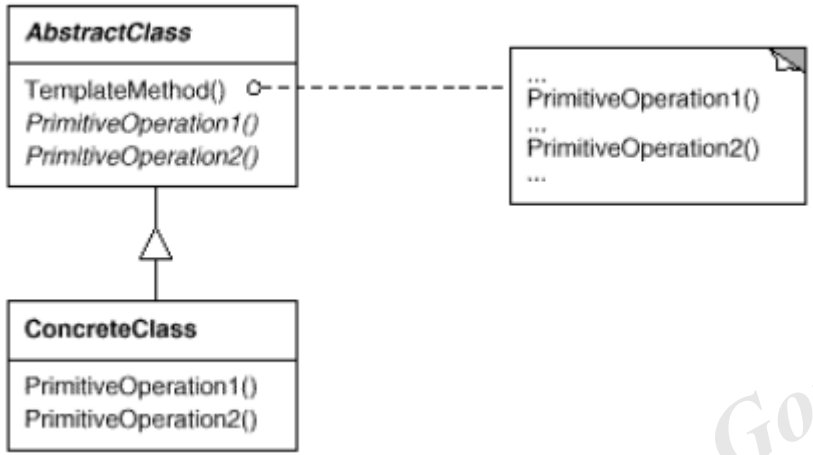
Padrão de Projeto Método-gabarito



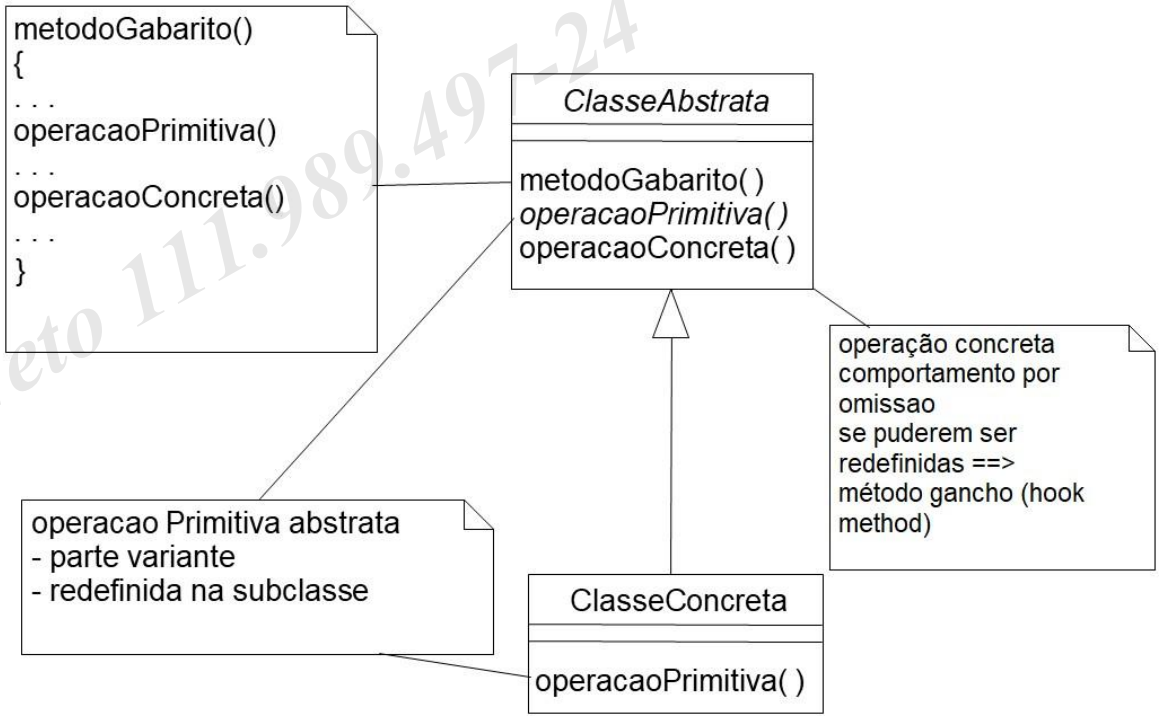
(Shvets, Alexander, 2018)

Padrão de Projeto Método-gabarito

- **Estrutura:**



*** Imagem capturada no livro versão html publicado em CD (Gamma et al 1995)*

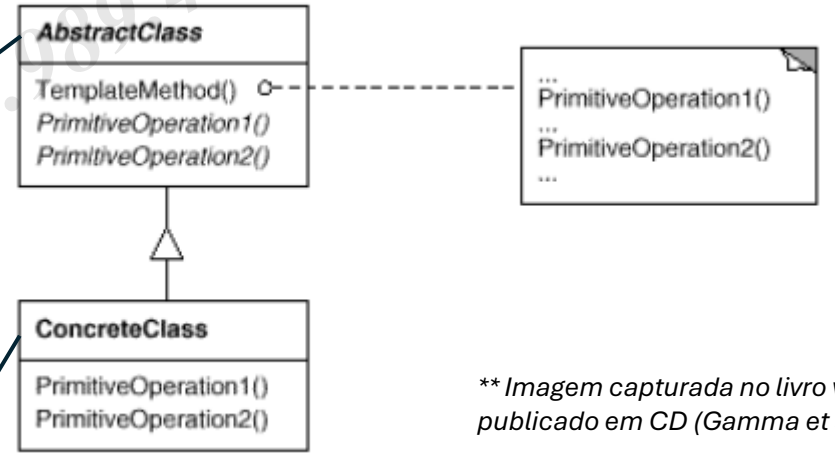


(Braude, 2005)

Padrão de Projeto Método-gabarito

- **Participantes:**

- define operações primitivas abstratas que subclasses concretas implementam como etapas de um algoritmo.
- implementa um método-gabarito que define o esqueleto de um algoritmo. Ele chama operações primitivas, bem como operações definidas na classe abstrata ou até em outros objetos.



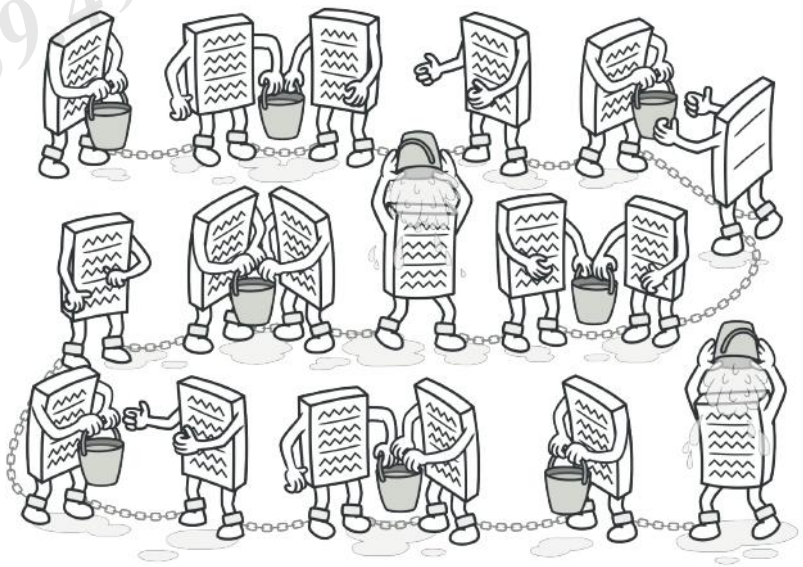
*** Imagem capturada no livro versão html publicado em CD (Gamma et al 1995)*

- implementa as operações primitivas para executar etapas específicas da subclasse do algoritmo.

Enquete 9:
Padrão relacionado?

Padrão de Projeto Cadeia de Responsabilidades (do inglês Chain of Responsibilities)

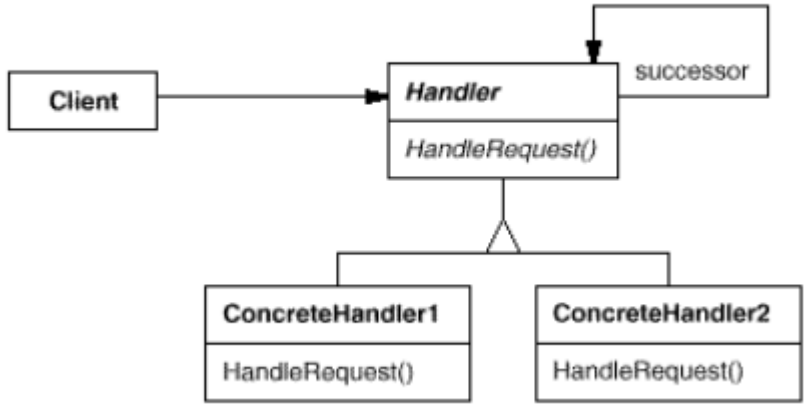
- **Classificação:** Padrão comportamental/Escopo de objetos
- **Intenção**: Evitar acoplar o remetente de uma solicitação ao seu destinatário dando a mais de um objeto a chance de manipular a solicitação. Encadear os objetos receptores e passar a solicitação ao longo da cadeia até que um objeto a manipule.
- **Aplicabilidade**: use o padrão Cadeia de Responsabilidades quando seu programa precisa processar diferentes tipos de requisições de diversas formas, mas os tipos exatos de requisições e sua sequência não são conhecidas antecipadamente



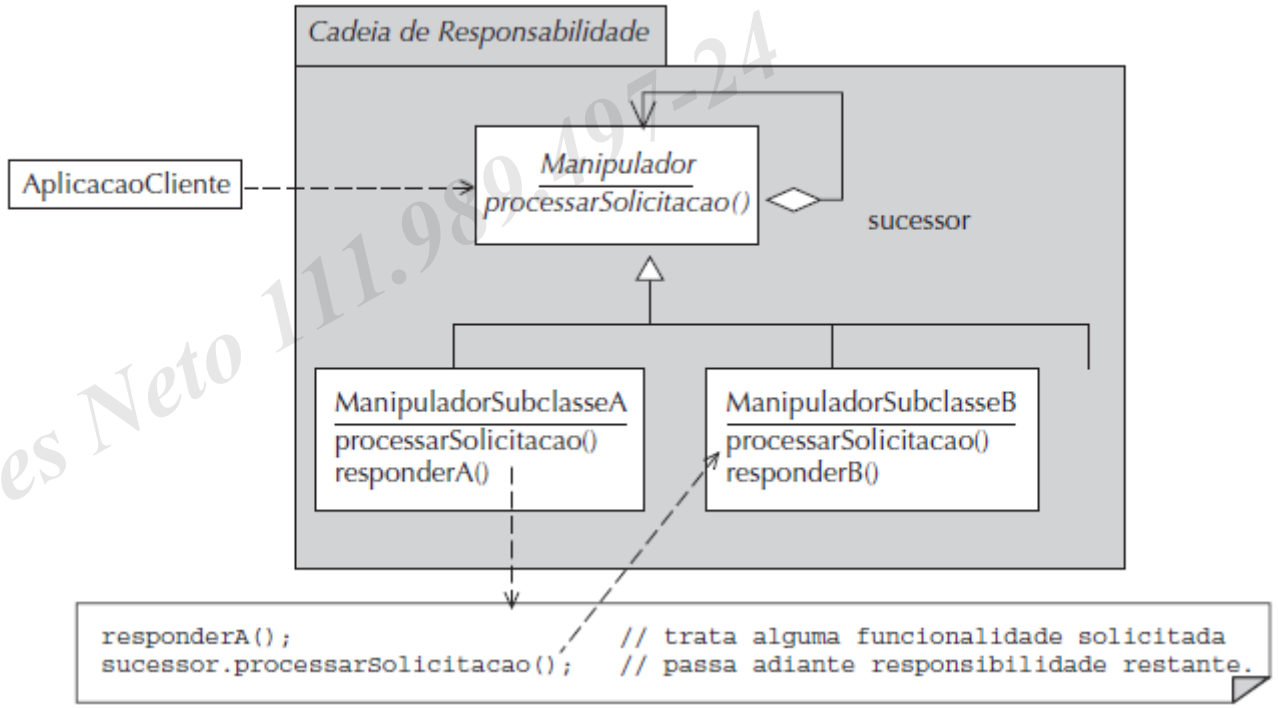
(Shvets, Alexander, 2018)

Padrão de Projeto Cadeia de Responsabilidades

- **Estrutura:**



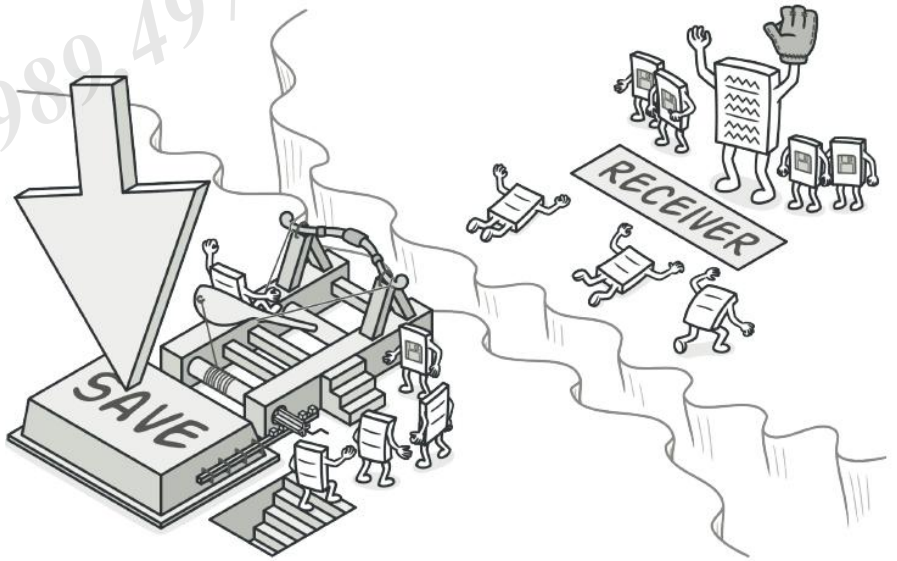
*** Imagem capturada no livro versão html publicado em CD (Gamma et al 1995)*



(Braude, 2005)

Padrão de Projeto Comando (do inglês Command)

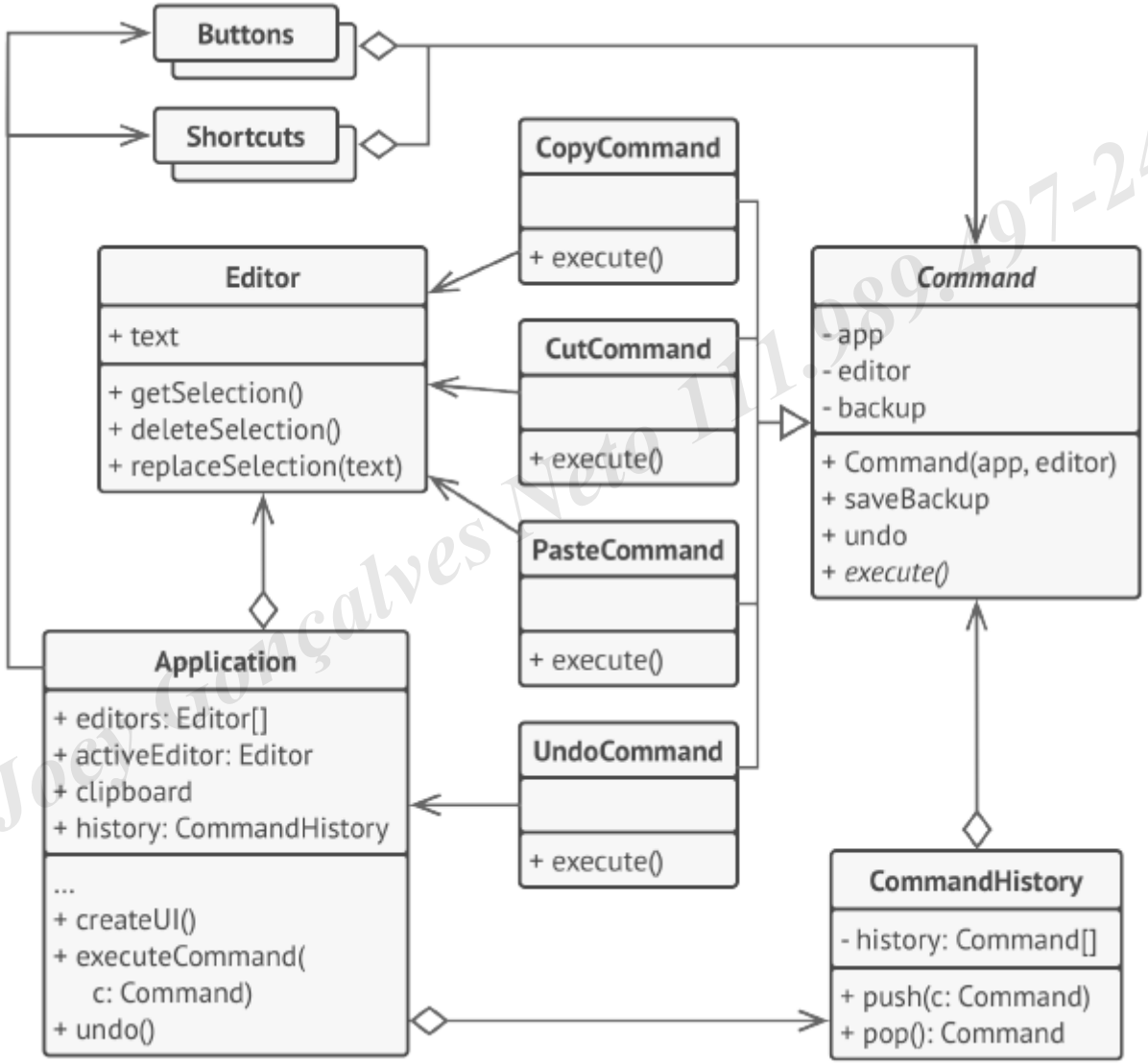
- **Classificação:** Padrão comportamental/Escopo de objetos
- **Intenção**: Encapsular uma solicitação como um objeto, permitindo assim parametrizar os clientes com diferentes solicitações, criar filas ou logs de solicitações e dar suporte a operações que podem ser desfeitas.
- **Motivação**: em algumas aplicações, você pode precisar parametrizar objetos com operações.



(Shvets, Alexander, 2018)

Padrão de Projeto Comando

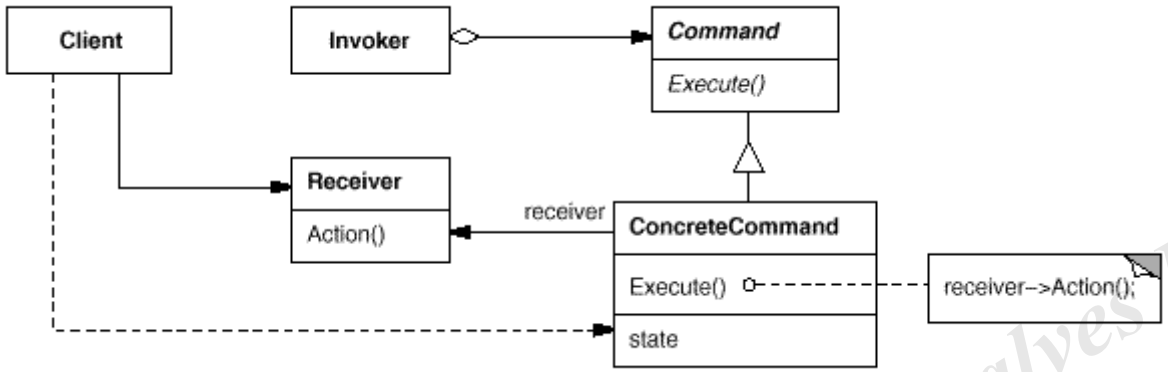
- **Exemplo:**



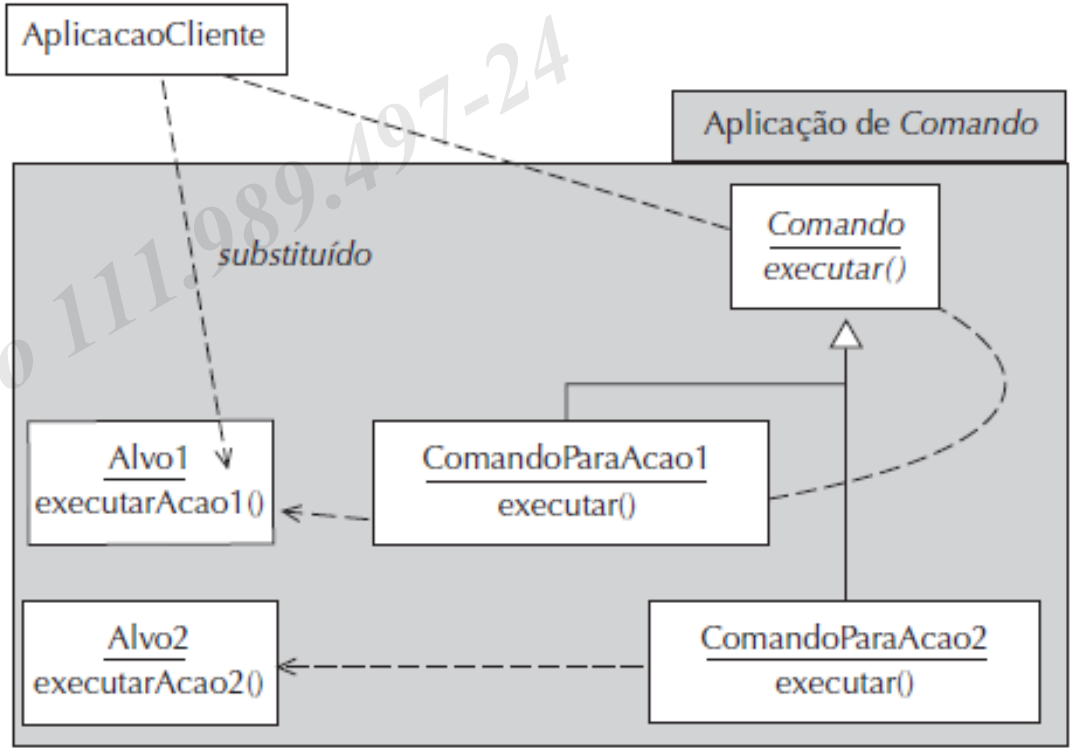
(Shvets, Alexander, 2018)

Padrão de Projeto Comando

- **Estrutura:**



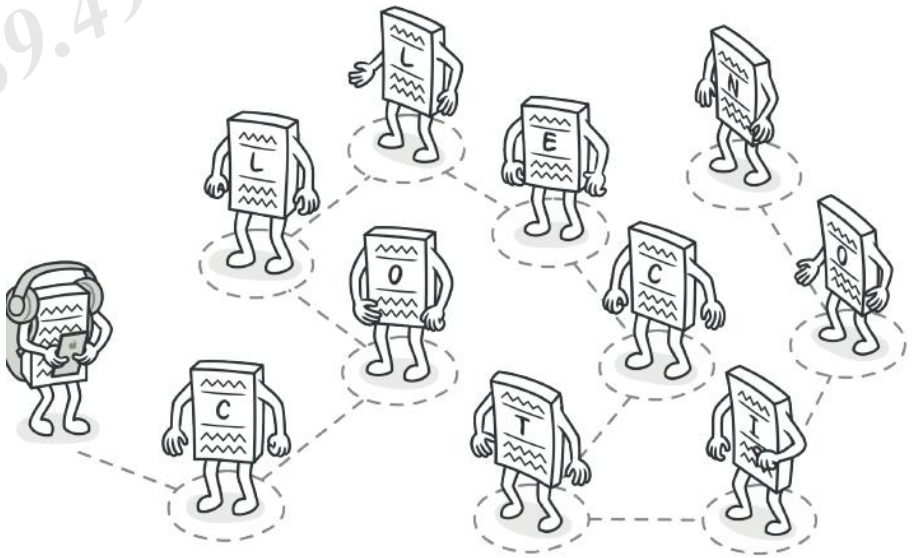
*** Imagem capturada no livro versão html publicado em CD (Gamma et al 1995)*



(Braude, 2005)

Padrão de Projeto Iterador (do inglês Iterator)

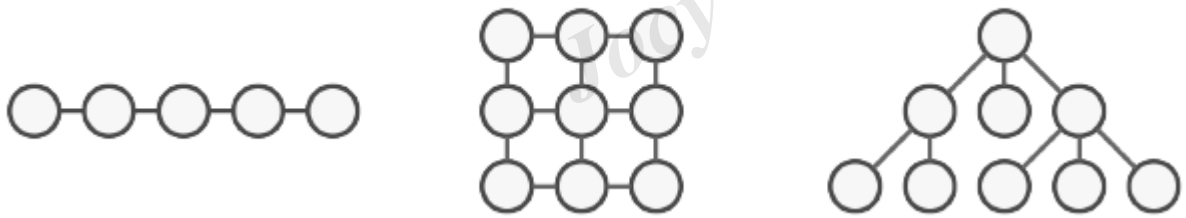
- **Classificação:** Padrão comportamental/Escopo de objetos
- **Intenção**: Fornecer uma maneira de acessar os elementos de um objeto agregado sequencialmente sem expor sua representação subjacente.
- **Motivação**: Existe a necessidade de percorrer agregados quaisquer, em diversas ordens, mas cada um tem uma forma diferente de ser percorrido. Incluir muitos métodos para cuidar da iteração pode poluir a interface. Se for desejável percorrer o agregado de diferentes formas, ainda mais métodos serão necessários.

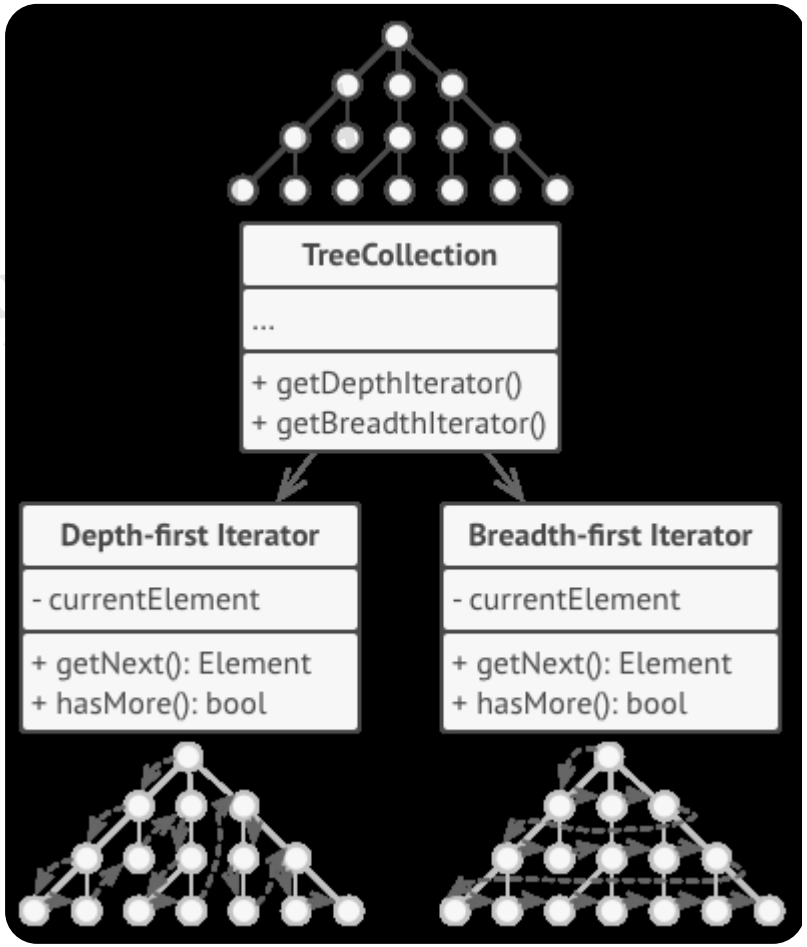


(Shvets, Alexander, 2018)

Padrão de Projeto Iterador

- **Aplicabilidade:**
- Use o padrão Iterador para:
  - acessar o conteúdo de um objeto agregado sem expor sua representação interna.
  - dar suporte a múltiplos trajetos de objetos agregados.
  - fornecer uma interface uniforme para percorrer diferentes estruturas agregadas (isto é, para apoiar iteração polimórfica)

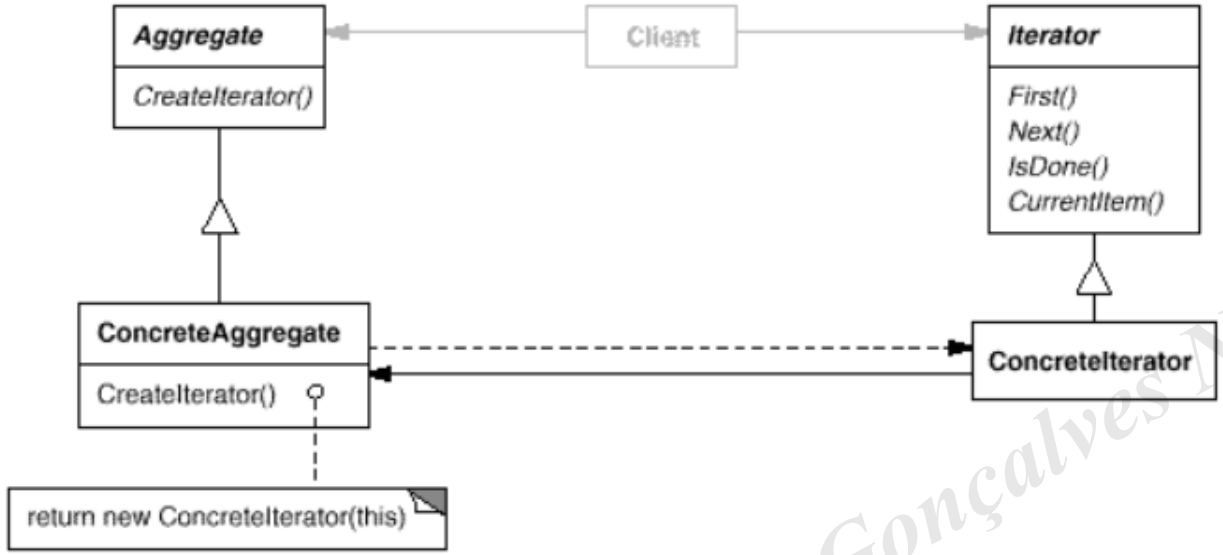




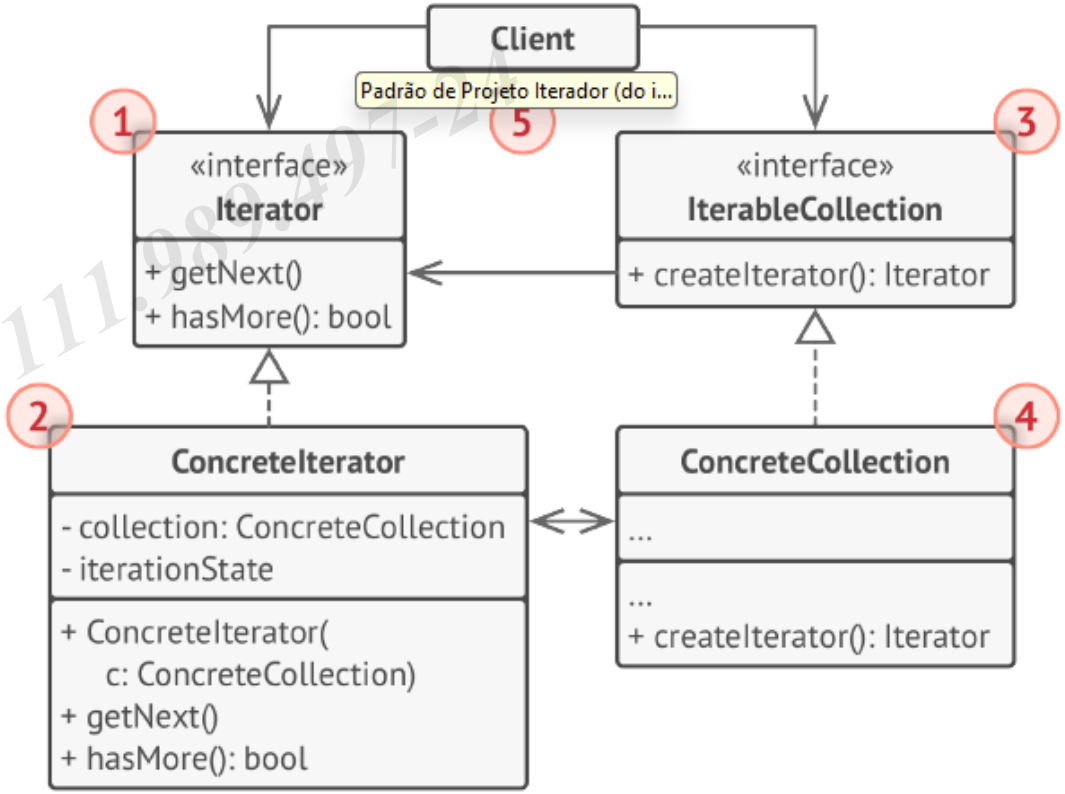
(Shvets, Alexander, 2018)

Padrão de Projeto Iterador

- **Estrutura:**



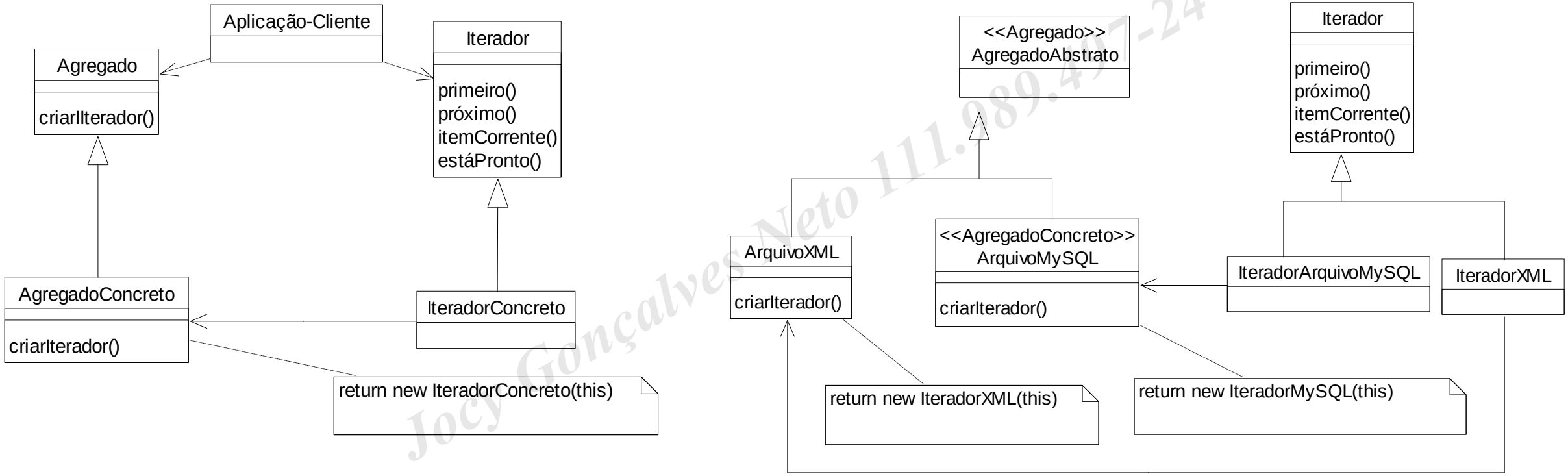
*** Imagem capturada no livro versão html publicado em CD (Gamma et al 1995)*



(Shvets, Alexander, 2018)

Padrão de Projeto Iterador

- **Exemplo:**

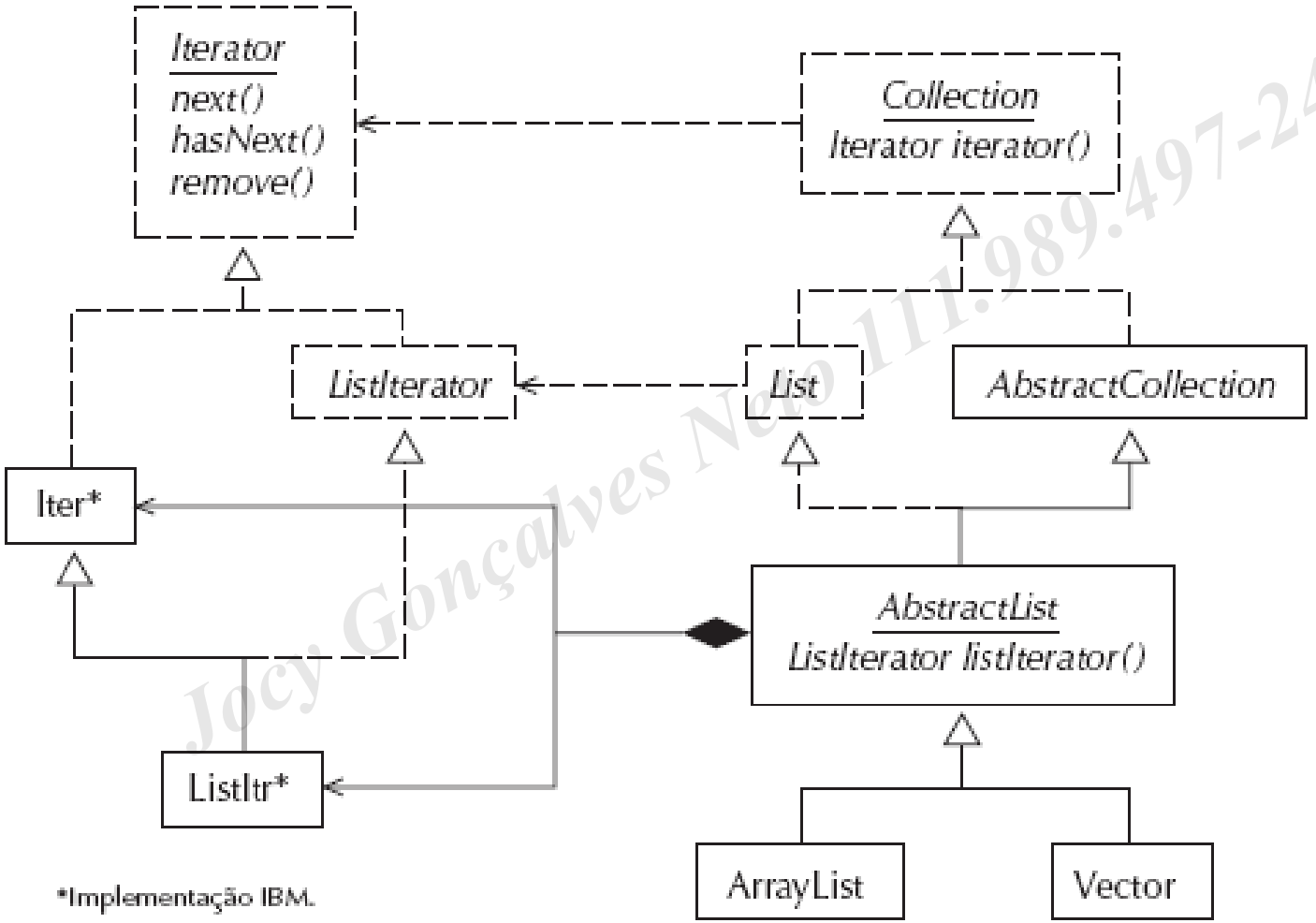


(Braude, 2005)

Padrão de Projeto Iterador

- **Iterador na API Java**
  - O pacote java.util contém uma interface Iterator com os métodos:next(), hasNext() e remove().
  - O método next() é uma combinação de proximo() e itemCorrente()
  - A subinterface ListIterator especifica esses métodos para operarem nos objetos List e adiciona os métodos apropriados aos objetos List.
  - A interface List tem um método listIterator(), que retorna um objeto ListIterator que itera sobre ele. Isso permite ao programador mover-se e iterar sobre objetos List.

Padrão de Projeto Iterador

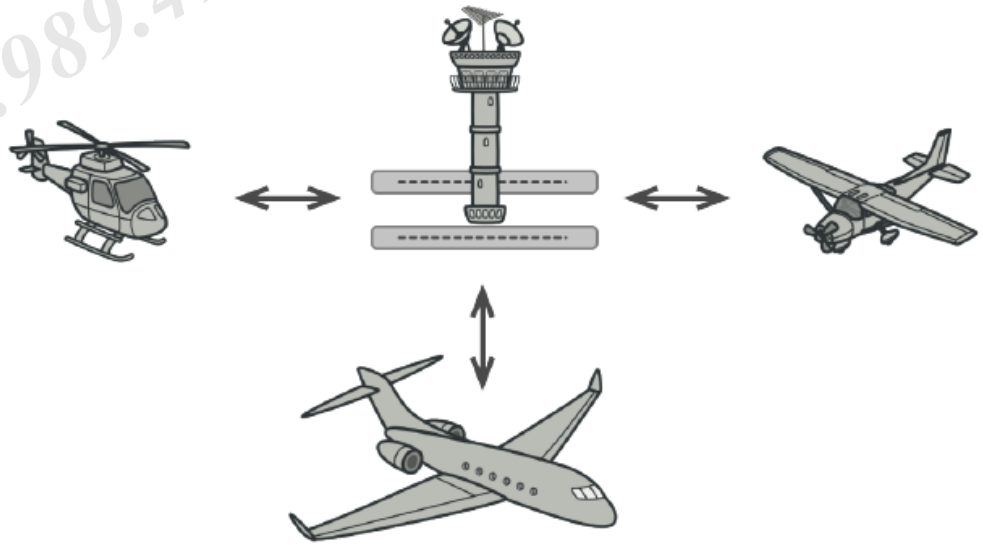


Padrão de Projeto Iterador

```
public class Leitor
{
  private String nome;
  private Char[] tipo;
  private Boolean achou=false;
  private List emprestimos = new ArrayList();
  public void devolverCopia(int codCopia)
   {
    Iterator i = emprestimos.iterator();
    while (i.hasNext()) && (!achou) {
       Emprestimo e = (Emprestimo) i.next();
       achou=e.devolverCopia(codCopia)}
   }
}
```

Padrão de Projeto Mediador (do inglês Mediator)

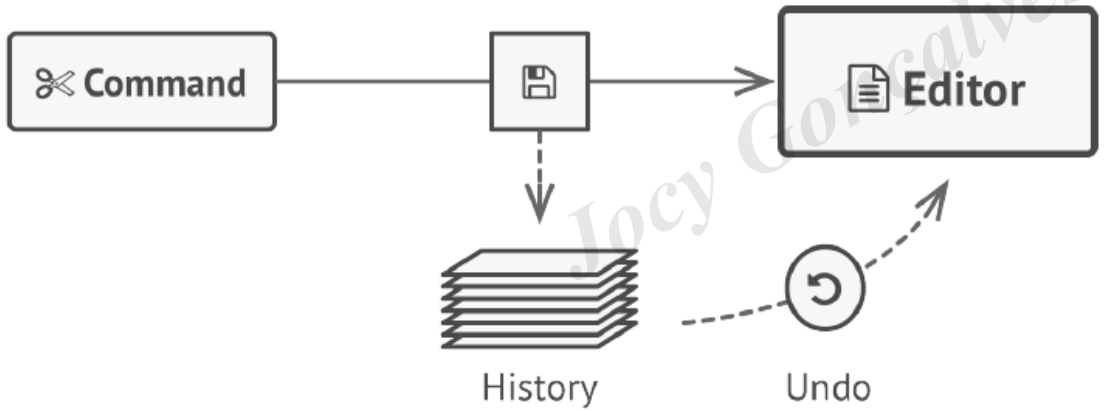
- **Classificação:** Padrão comportamental/Escopo de objetos
- **Intenção**: Definir um objeto que encapsula a forma como um conjunto de objetos interage. O Mediador promove acoplamento fraco ao impedir que objetos se refiram uns aos outros explicitamente, e permite que você varie a interação deles independentemente.



(Shvets, Alexander, 2018)

Padrão de Projeto Memento

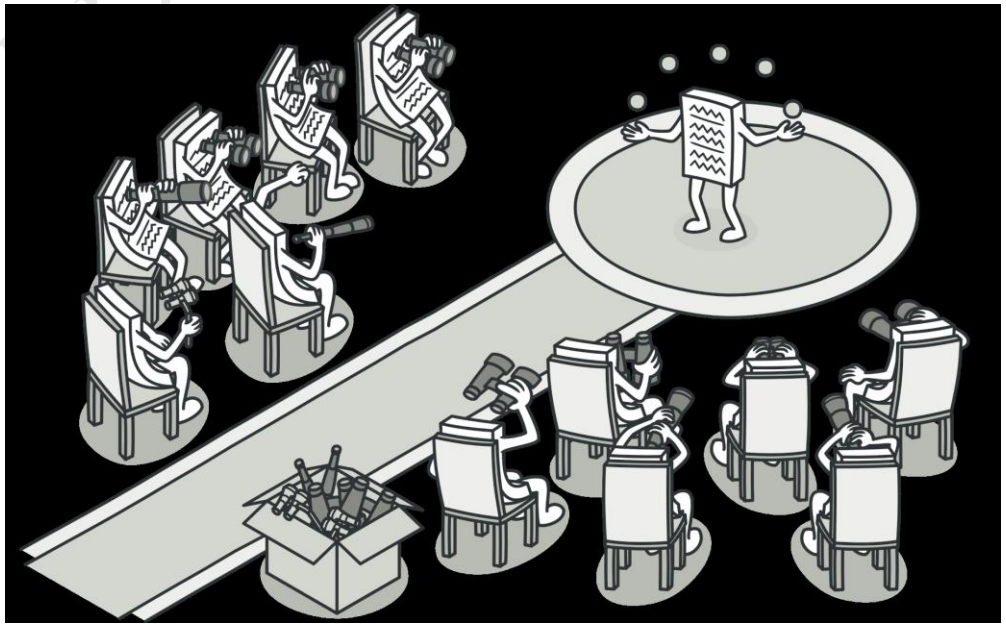
- **Classificação:** Padrão comportamental/Escopo de objetos
- **Intenção**: Sem violar o encapsulamento, capturar e externalizar o estado interno de um objeto para que o objeto possa ser restaurado a esse estado posteriormente.



(Shvets, Alexander, 2018)

Padrão de Projeto Observador (em inglês Observer)

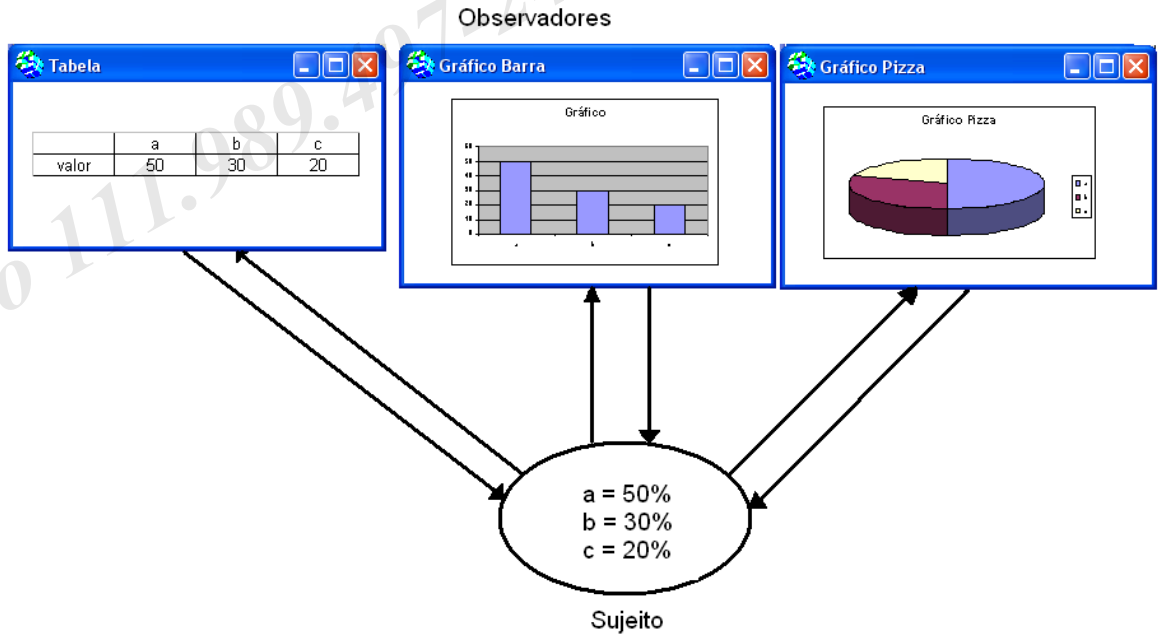
- **Classificação:** Padrão comportamental/Escopo de objetos
- **Intenção**: Definir uma dependência de um para muitos entre objetos para que, quando um objeto mudar de estado, todos os seus dependentes sejam notificados e atualizados automaticamente.



(Shvets, Alexander, 2018)

Padrão de Projeto Observador

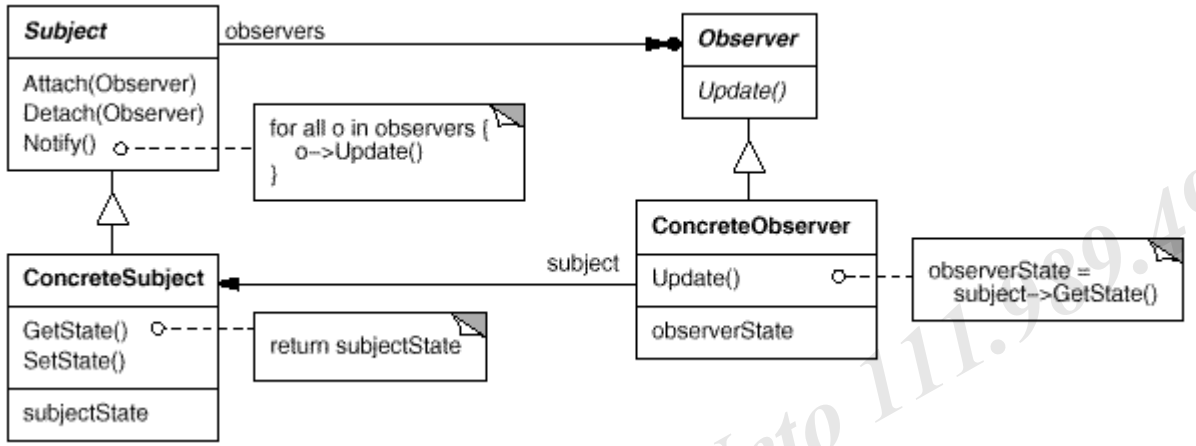
- **Aplicabilidade:**
  - sistemas cujas informações são preferencialmente centralizadas em uma única fonte (sujeito), e existem diversas visões (observadores) que devem ser atualizadas sempre que a informação for modificada
    - Qualquer tentativa de manter a consistência aumenta o acoplamento entre as classes.
    - Por exemplo, em uma ferramenta integrada, pode-se separar os dados em si das diversas visões sobre esses dados, de forma que mudança em uma das visões seja refletida nas demais, mesmo que as visões não tenham conhecimento umas das outras.
  - Quando um objeto deve ser capaz de notificar outros objetos sem fazer suposições sobre quem são esses objetos, ou não os conhecendo de antemão ou quando esses objetos observadores mudam dinamicamente



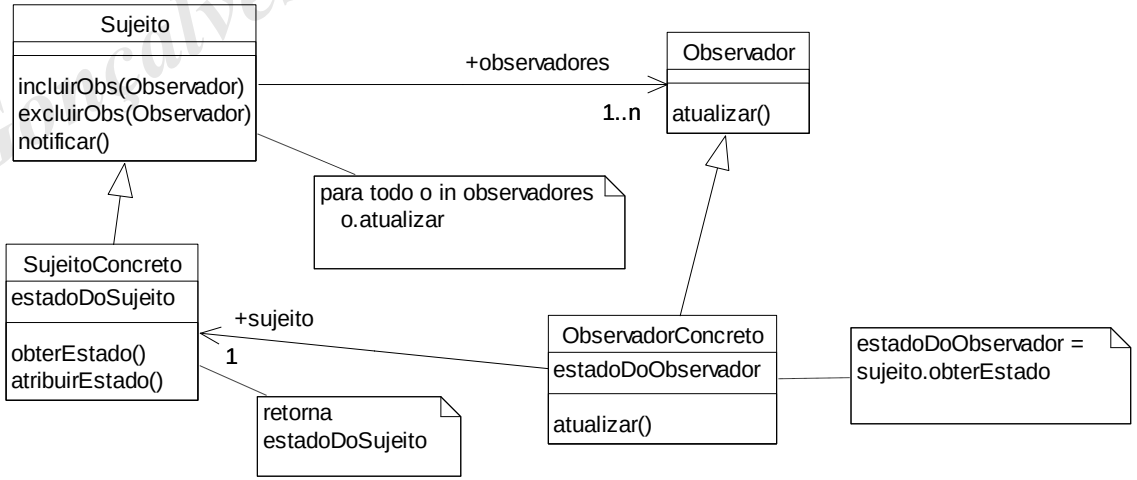
Enquete 10:
O que isso te lembra?

Padrão de Projeto Observador

- **Estrutura:**



*** Imagem capturada no livro versão html publicado em CD (Gamma et al 1995)*

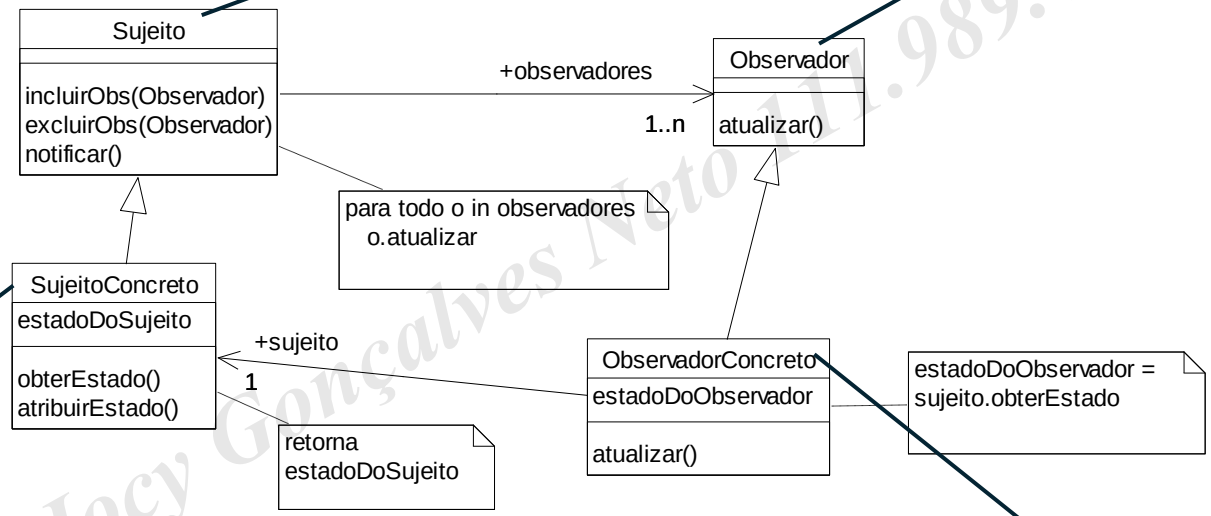


Padrão de Projeto Observador

- **Participantes:**

Conhece seus observadores, provê uma interface para incluir e exclui-los

Define uma interface de atualização para objetos que precisarão ser notificados de mudanças no sujeito

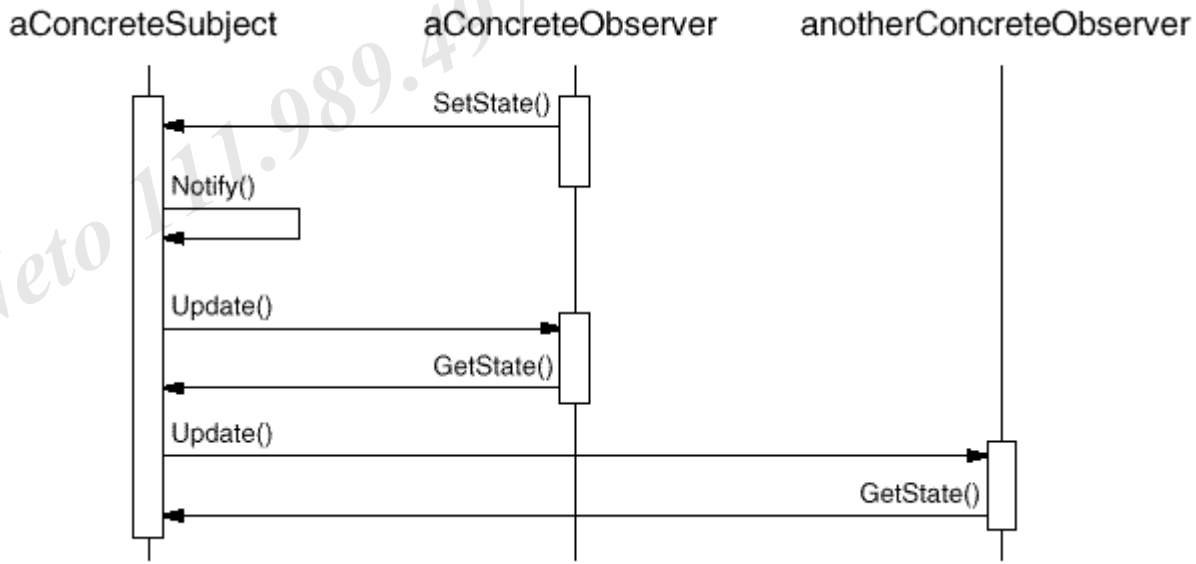


- Armazena o estado de interesse para objetos ObservadorConcreto.
- Envia uma notificação para seus observadores quando seu estado muda

- Mantem referência para o objeto SujeitoConcreto
- Armazena o estado que deve ser consistente com o sujeito
- Implementa o método abstrato atualizar

Padrão de Projeto Observador

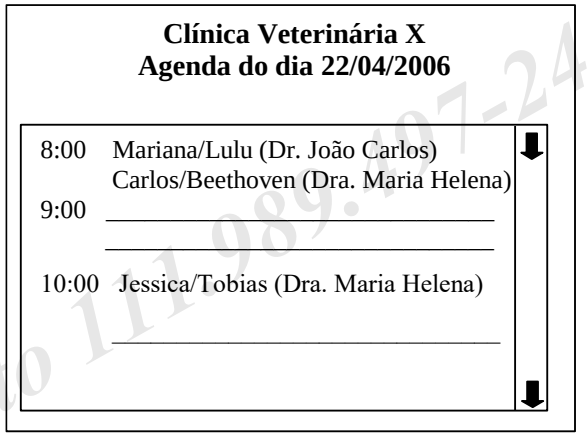
- **Formas de implementação:**
  - Os próprios observadores podem enviar atualizações de estado para o sujeito concreto, que por sua vez notifica a todos os observadores
  - Os demais observadores podem obter o estado para saber se precisam se atualizar ou não
  - Outra forma de implementação seria os observadores, ao se inscrever, especificar algumas condições para que só recebam a notificação caso as condições sejam verdadeiras
  - No livro de Gamma há várias outras considerações importantes para levar em conta ao implementar esse padrão

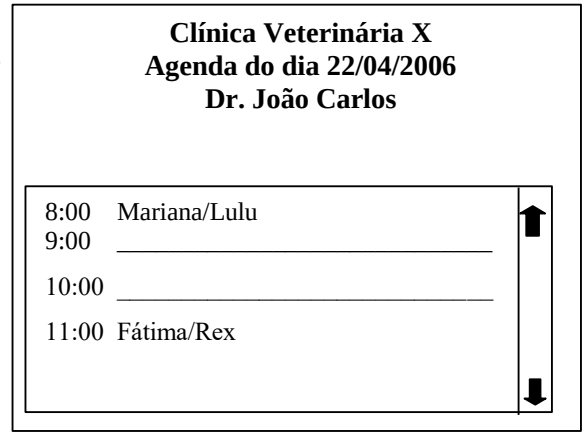


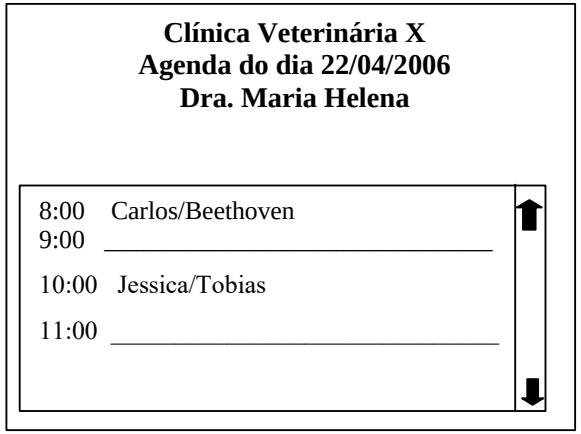
*** Imagem capturada no livro versão html publicado em CD (Gamma et al 1995)*

Padrão de Projeto Observador

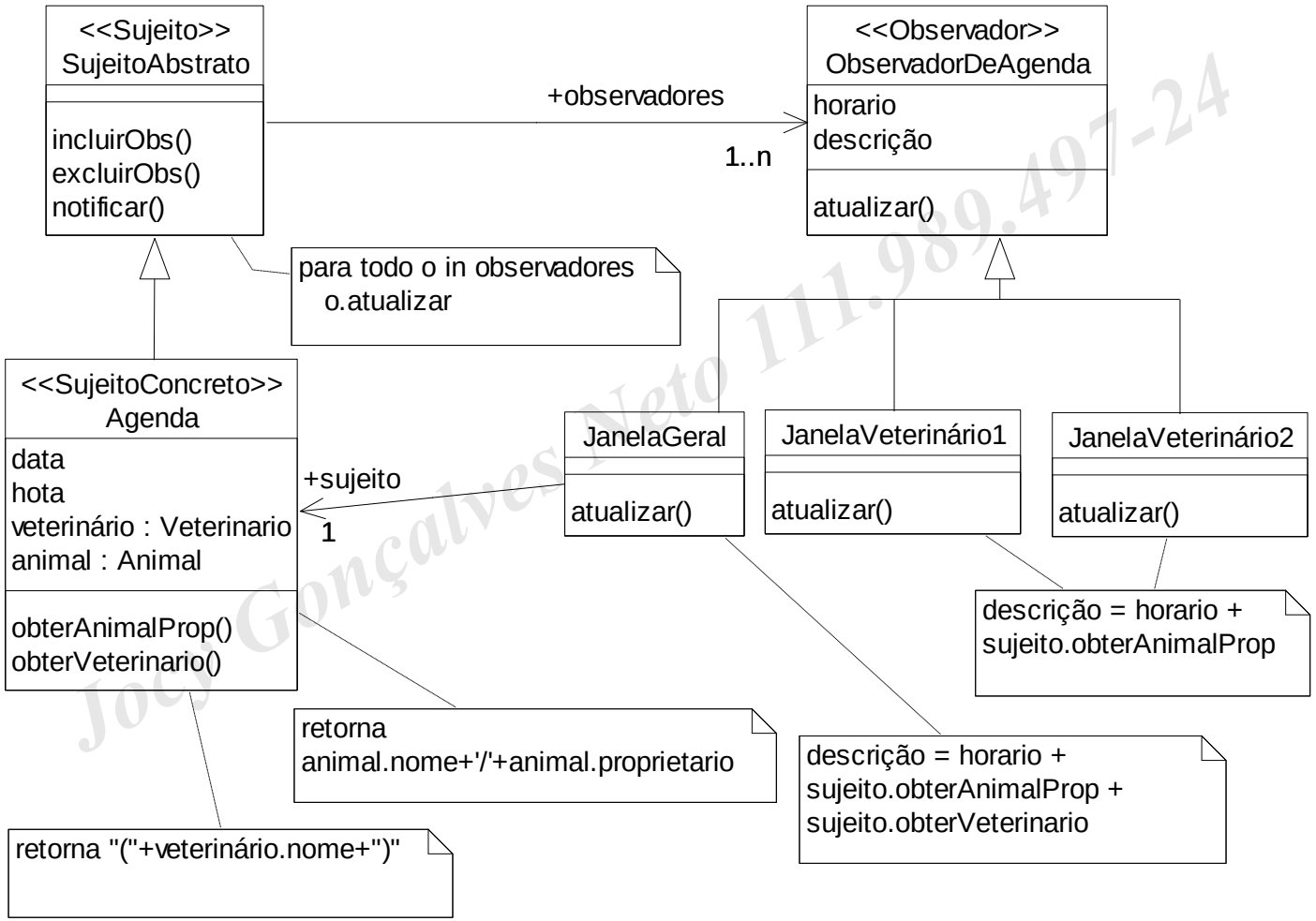
- **Exemplo:**
- projetar a interface gráfica com o usuário para um sistema de agendamentos atribuídos aos diversos veterinários que trabalham em uma clínica.
- Na visão da secretária, devem aparecer todos os agendamentos
- Na visão de cada veterinário, aparece somente sua agenda particular





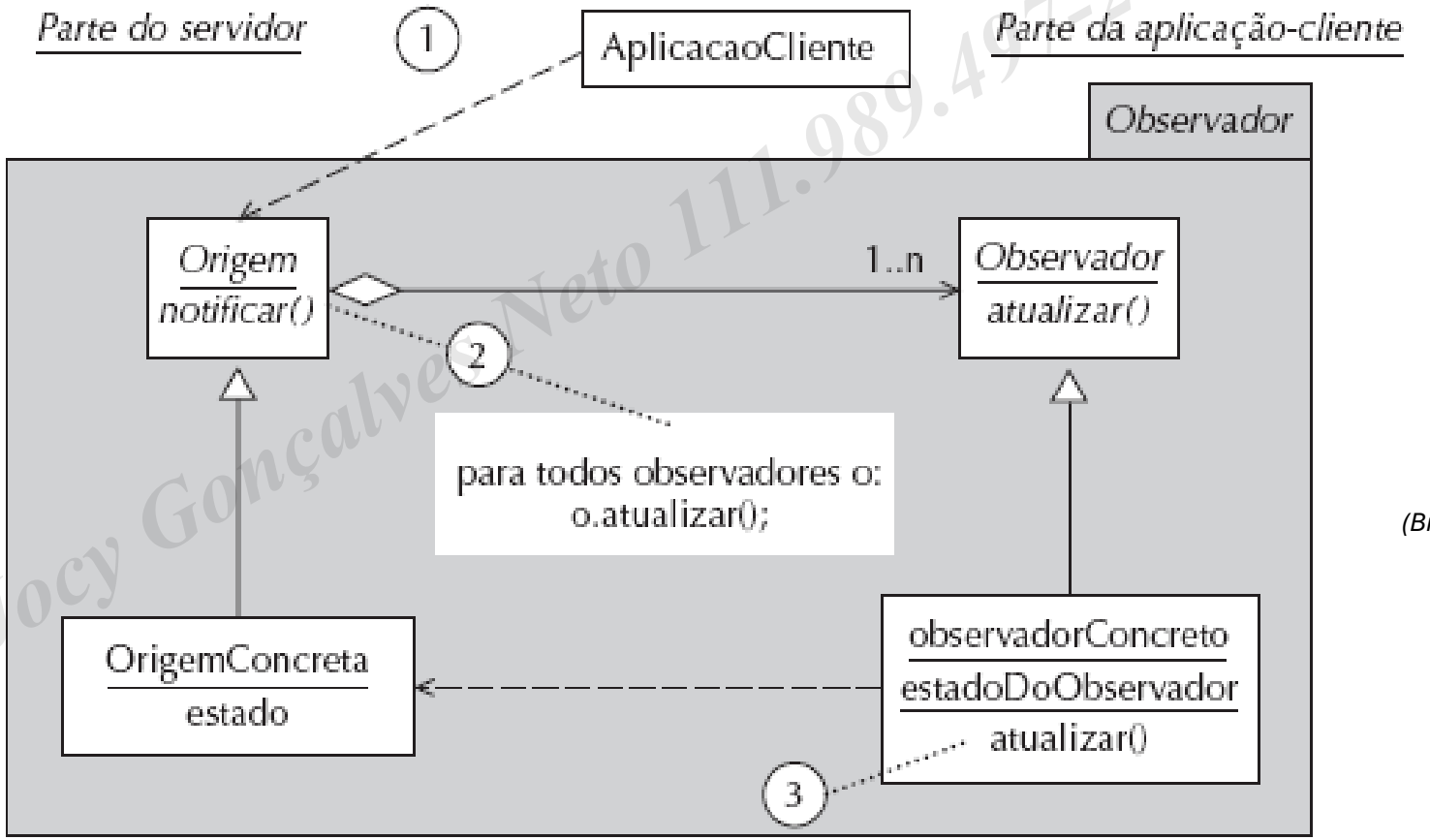


Padrão de Projeto Observador



Padrão de Projeto Observador

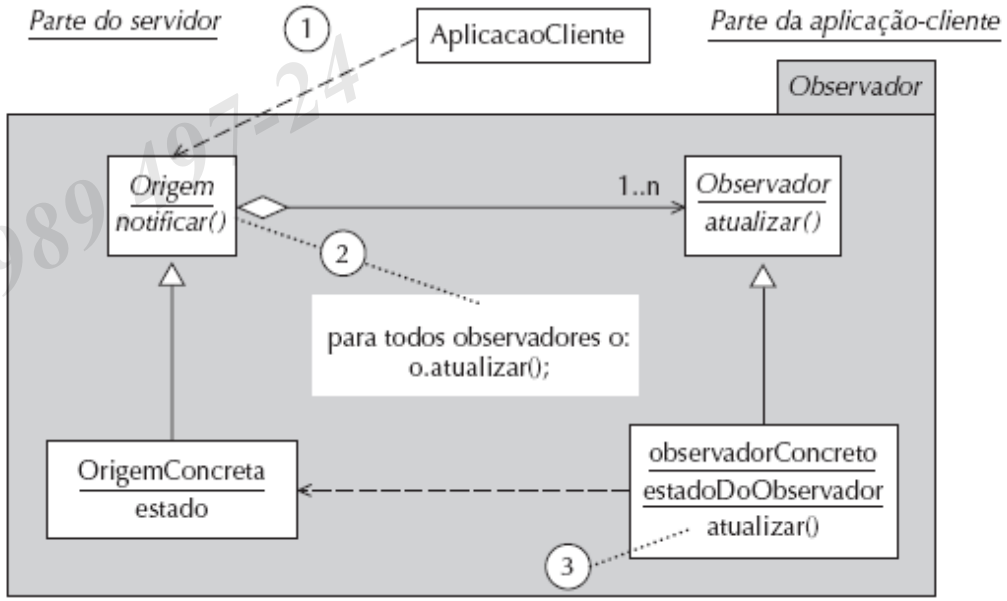
- Como funciona o Observador?



(Braude, 2005)

Padrão de Projeto Observador

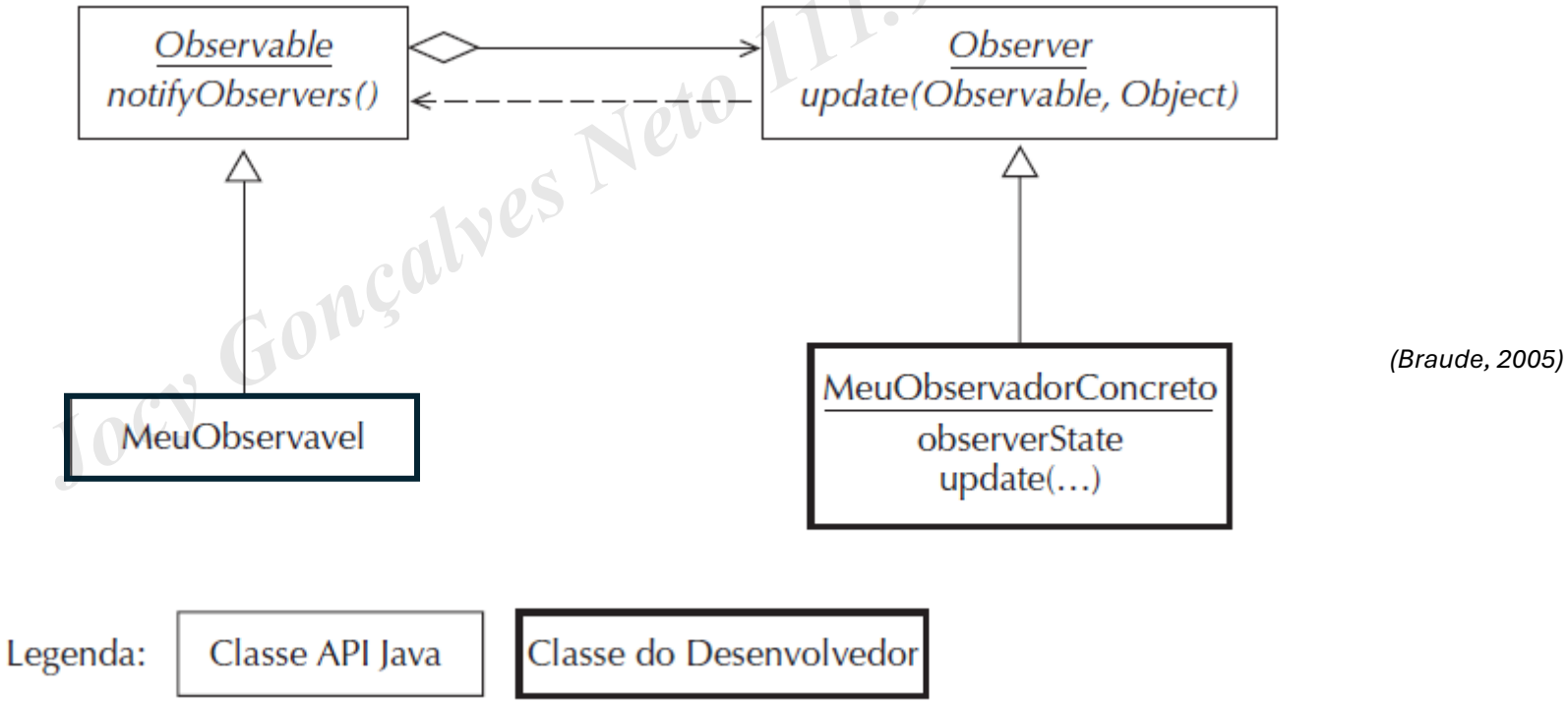
- **Como funciona?**
  - (Passo 1) A aplicação cliente referencia um objeto de interface conhecido, solicitando que os observadores sejam notificados.
    - Por exemplo, a aplicação cliente poderia ser um processo programado para alertar sobre uma alteração de dados. No modelo, isso é mostrado como um objeto AplicacaoCliente, que informa ao objeto Origem para executar sua função notificar().
  - (Passo 2) O método notificar() chama a função atualizar() em cada objeto Observador que ele agrega.
  - (Passo 3) A implementação de atualizar() depende do ObservadorConcreto particular a que pertence.
    - Normalmente, atualizar() compara o estado do objeto ObservadorConcreto (valores de variáveis) com aquele da origem de dados central, para então decidir se deve ou não alterar seus valores de variáveis da mesma forma.



(Braude, 2005)

Padrão de Projeto Observador

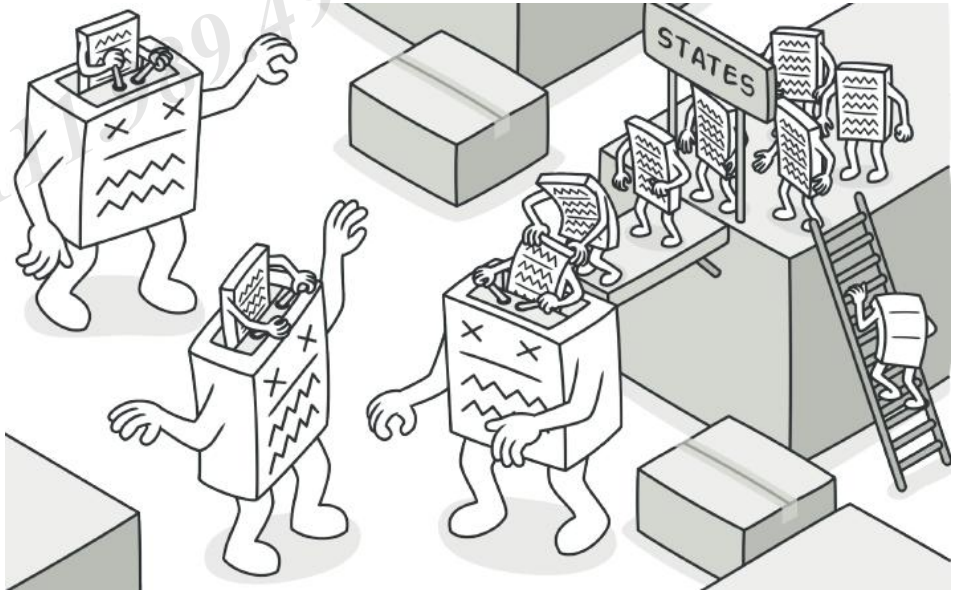
- Vale lembrar que esse padrão existe na API Java
  - Observe que update() é um método call-back, porque fornece aos objetos Observer uma referência a sua origem, permitindo que comparem seus dados, por exemplo com o objeto Observable ao executar o update.
  - Como a atualização é feita de modo retroativo, não há necessidade das classes concretas Observer manterem referências ao objeto Observable.



(Braude, 2005)

Padrão de Projeto Estado (do inglês State)

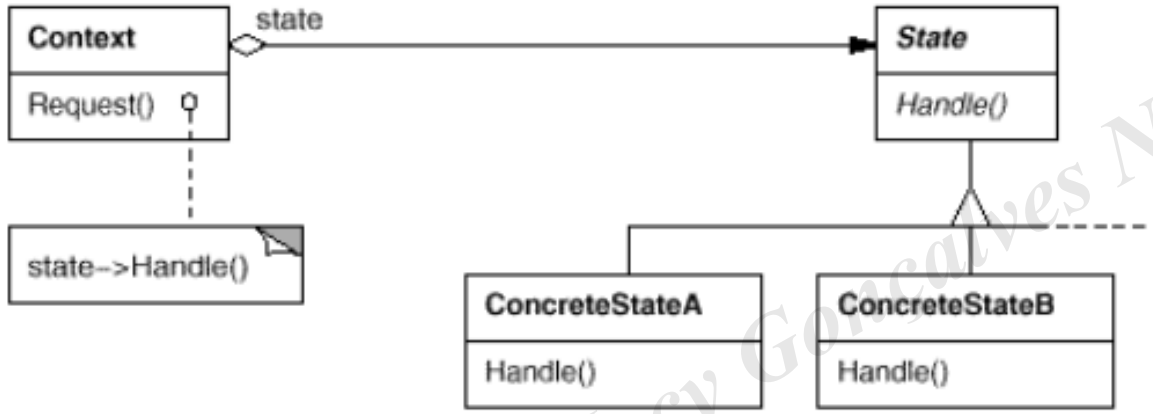
- **Classificação:** Padrão comportamental/Escopo de objetos
- **Intenção**: Permitir que um objeto altere seu comportamento quando seu estado interno mudar. A ideia é fazer parecer que o objeto mudou de classe.
- **Aplicabilidade**:
  - quando o comportamento de um objeto depende de seu estado, que pode mudar em tempo de execução
  - quando essa mudança de comportamento implementada com o uso de estruturas case favorece erros, principalmente quando o sistema evolui e novos estados surgem, pois várias operações conterão essas estruturas case.



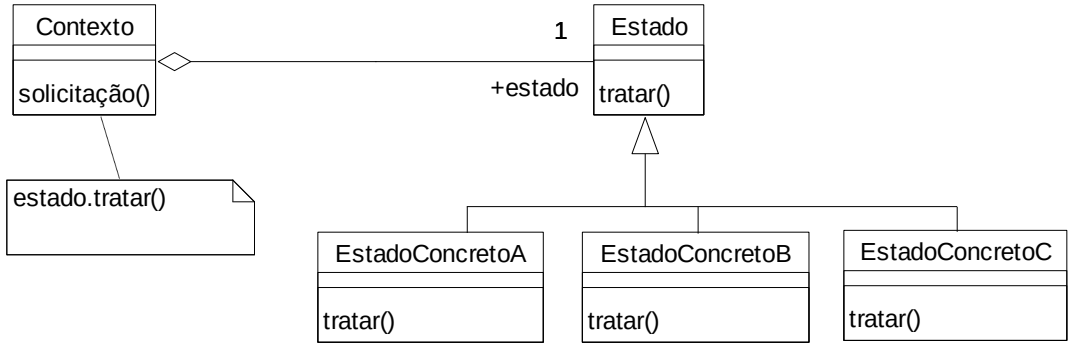
(Shvets, Alexander, 2018)

Padrão de Projeto Estado

- **Estrutura:**
  - Criar uma classe para representar o objeto em si e outra para representar seu estado
  - O estado pode variar independentemente de outros objetos.

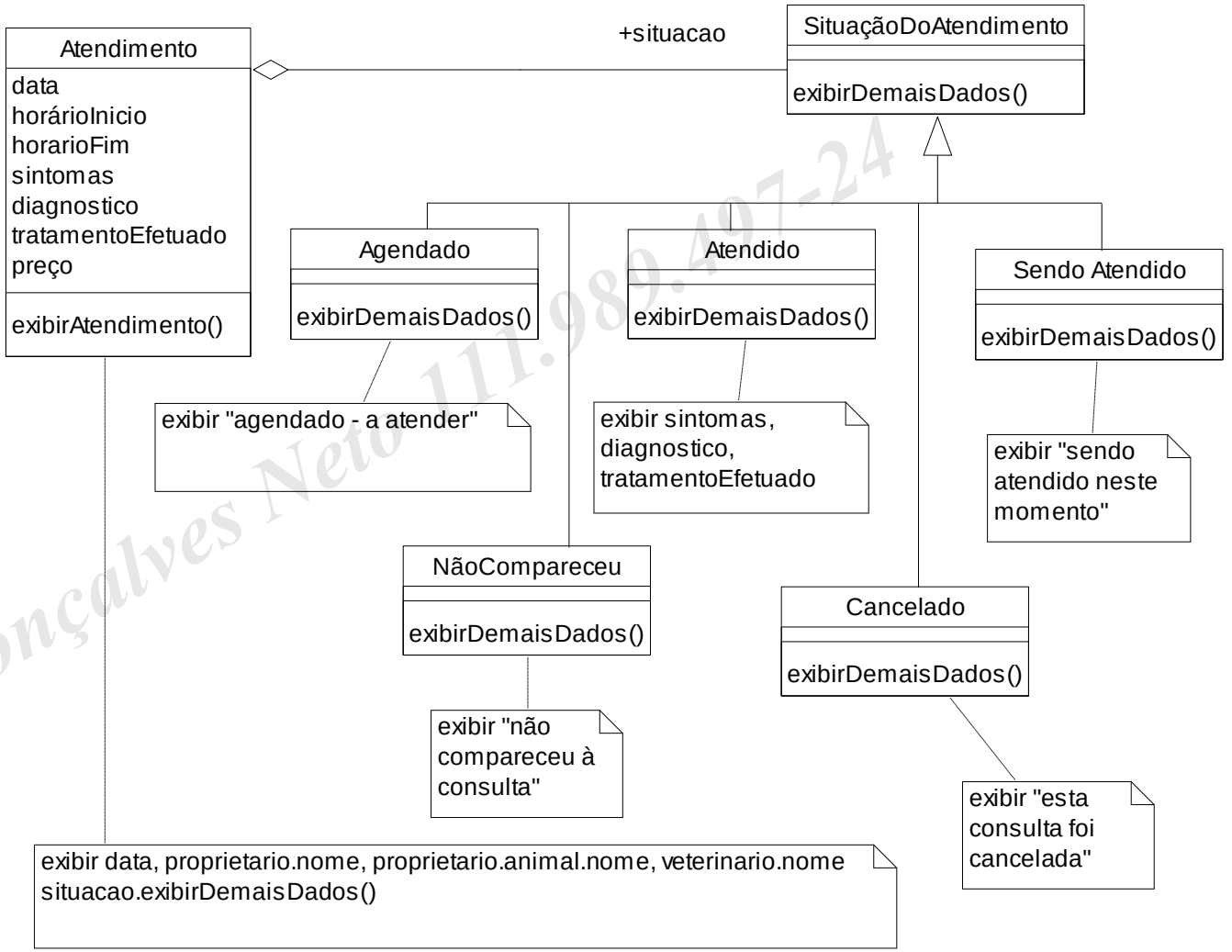


*** Imagem capturada no livro versão html publicado em CD (Gamma et al 1995)*



Padrão de Projeto Estado

- **Exemplo:**
  - Exibir uma mensagem que tem uma parte fixa e outra parte que depende do estado do atendimento do paciente



Enquete 11:
Uso do padrão Estado

Padrão de Projeto Estado

- **Questões de implementação:**
  - Quem define as transições de estado: contexto ou estado?
    - Se há um critério fixo para mudança de estado: contexto
    - Mudanças constantes: estado
      - Necessária interface para que seja possível que objetos estados modifiquem explicitamente o estado atual do contexto
  - Tabela para mapear transições
  - Criar e destruir objetos Estado
    - Criar todos os objetos a priori ou criar/destruir sob demanda?

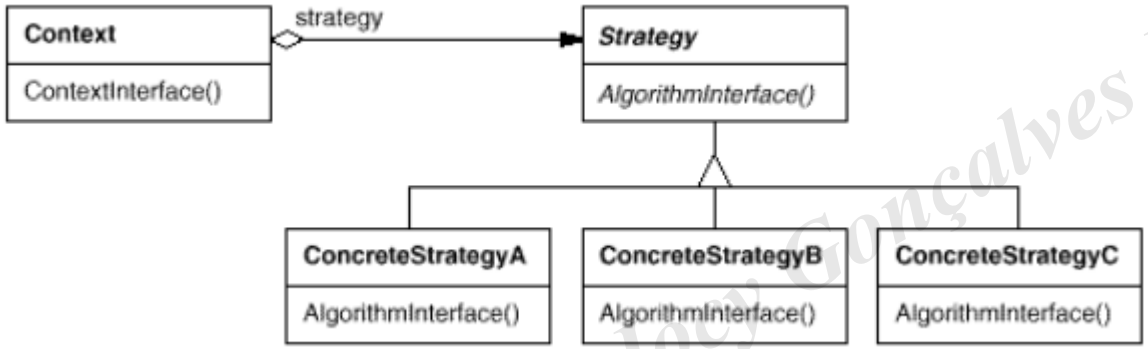
Padrão de Projeto Estratégia (do inglês Strategy)

- **Classificação:** Padrão comportamental/Escopo de objetos
- **Intenção**: Definir uma família de algoritmos, encapsular cada um dos algoritmos e torná-los intercambiáveis. A estratégia permite que o algoritmo varie independentemente dos clientes que o usam.
- **Aplicabilidade**:
  - várias classes relacionadas diferem apenas no comportamento ou
  - são necessárias diversas versões de um algoritmo ou
  - as aplicações-cliente não precisam saber detalhes específicos de estruturas de dados de cada algoritmo

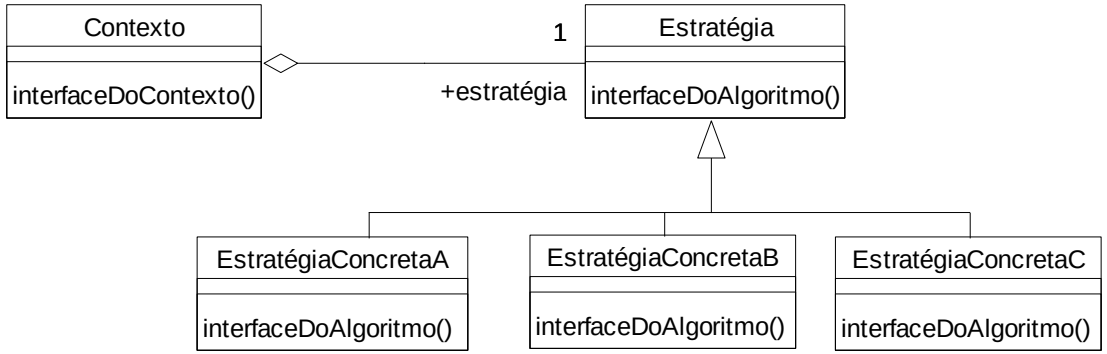
(Shvets, Alexander, 2018)

Padrão de Projeto Estratégia

- **Exemplo:**
  - Criar uma classe abstrata para a Estratégia empregada pelo algoritmo, bem como subclasses especializando cada um dos algoritmos.
  - O Contexto mantém uma referência para o objeto Estratégia e pode definir uma interface para permitir que a Estratégia acesse seus dados. A Estratégia define uma interface comum a todos os algoritmos disponíveis. O Contexto delega as solicitações recebidas das aplicações-cliente para sua estratégia.

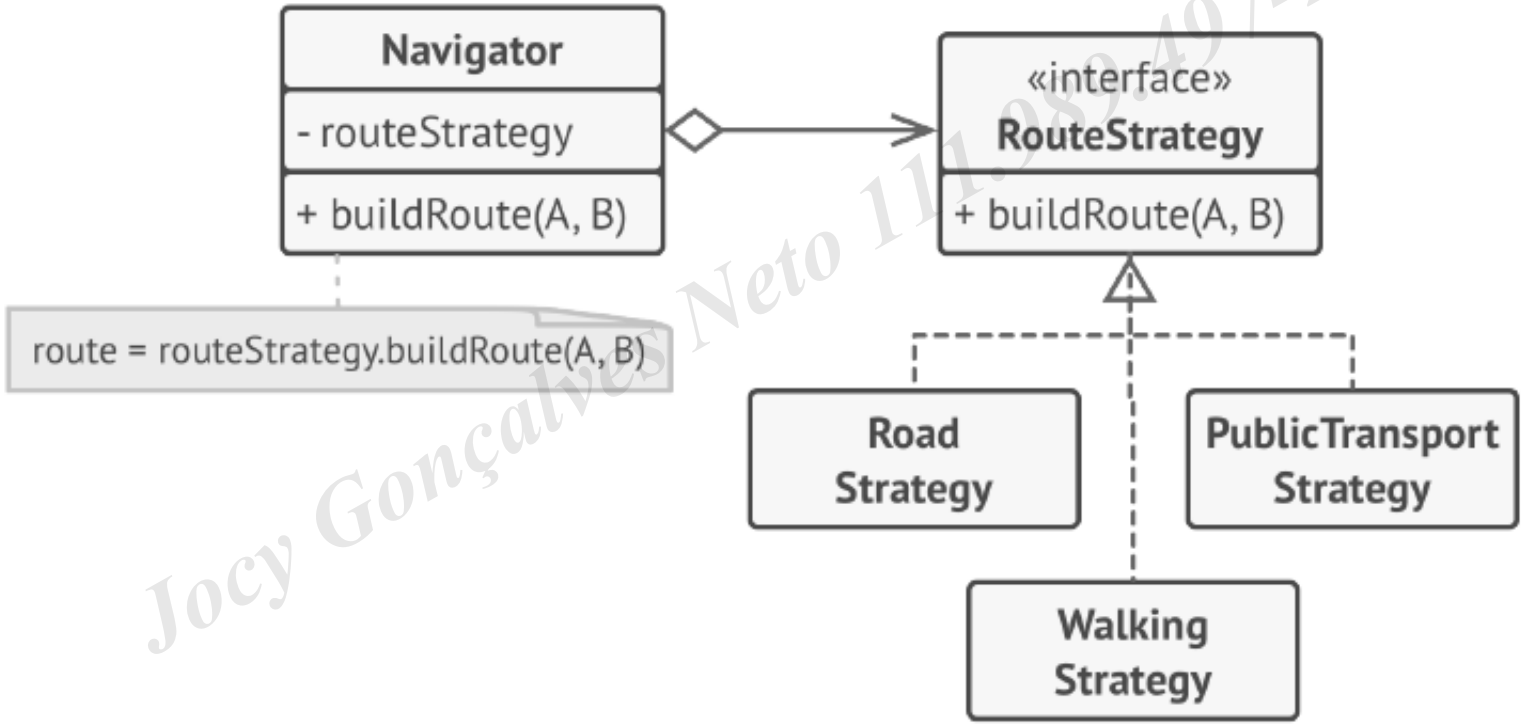


*** Imagem capturada no livro versão htm publicado em CD (Gamma et al 1995)*



Padrão de Projeto Estratégia

- **Exemplo:** Algoritmo para cálculo de rota depende do que o usuário escolher



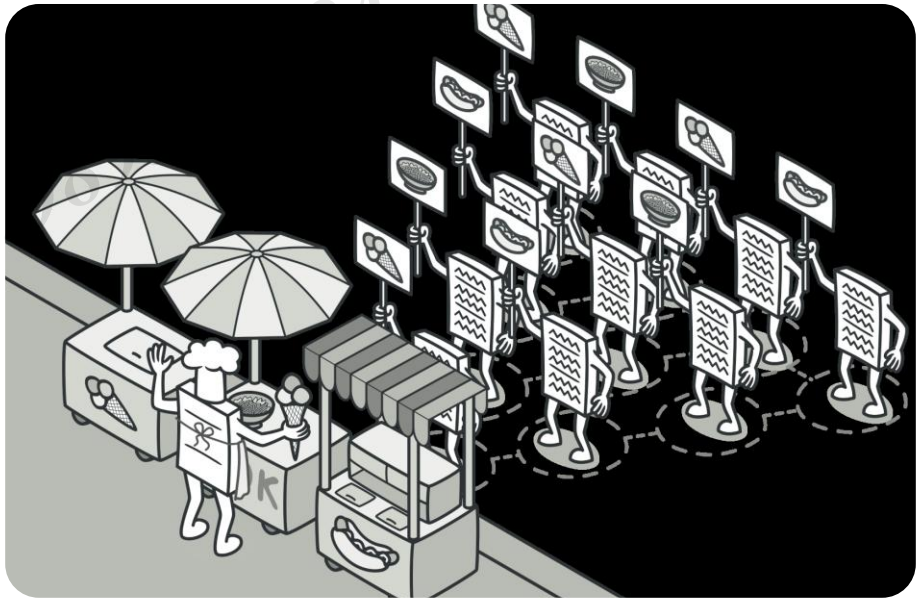
(Shvets, Alexander, 2018)

Diferença entre State e Strategy?

- State pode ser considerado uma extensão de Strategy.
- Ambos os padrões são baseados em composição: eles mudam o comportamento do contexto delegando algum trabalho para objetos auxiliares.
- Strategy torna os objetos completamente independentes e inconscientes dos outros.
- No entanto, State não restringe dependências entre estados concretos, permitindo que eles alterem o estado do contexto à vontade.

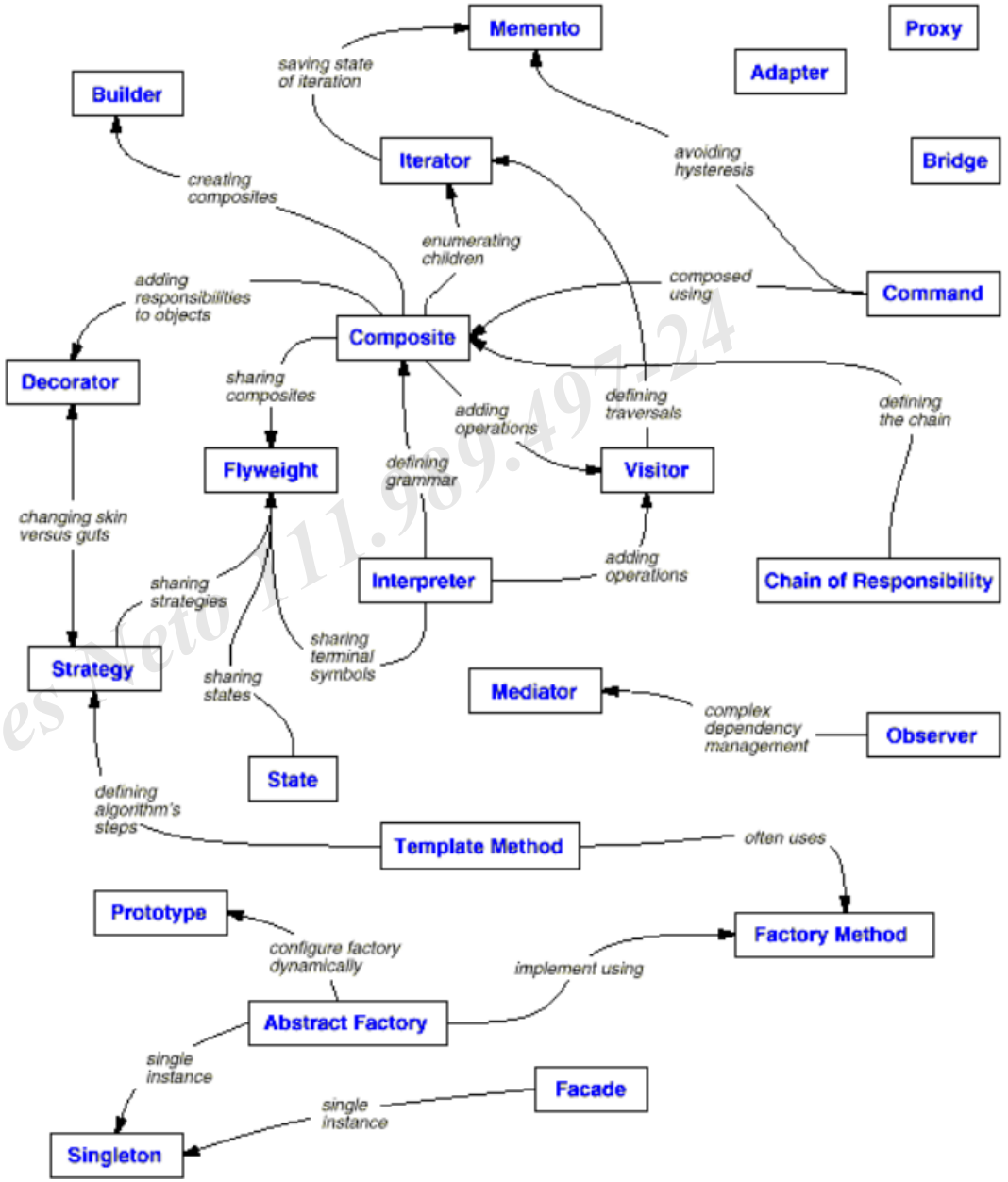
Padrão de Projeto Visitador (do inglês Visitor)

- **Classificação:** Padrão comportamental/Escopo de objetos
- **Intenção**: Permite que seja executada uma operação ao longo de um conjunto com objetos de diferentes classes, mas tendo um objeto "visitador" implementando variantes de uma mesma operação, que correspondem a todas as classes alvo.
  - Use esse padrão também quando um comportamento faz sentido apenas em algumas classes da hierarquia, mas não em outras



(Shvets, Alexander, 2018)

Relacionamentos entre os Padrões de Projeto



Conclusões

Vantagens:

- Soluções utilizadas em diversos projetos e que se provaram úteis
- Soluções com vantagens e desvantagens conhecidas
- Oferecem vocabulário comum entre desenvolvedores
- Aderem às melhores práticas de desenvolvimento OO
- Sugerem sistemas extensivos, flexíveis e robustos

Desvantagens:

- Criação de diversas classes e interfaces auxiliares, gerando mais complexidade
- Podem gerar overhead de memória e processamento
- Podem ser piores do que soluções simples quando utilizados em contextos errados

Referências

- Biggerstaff, Ted J. ; Perlis, Alan J. **Software Reusability: Concepts and Models**. Acm Press Frontier Series, 1989
- Alexander et al. **A Pattern Language**. New York: Oxford University Press, 1977.
- Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. **Design Patterns - Elements of Reusable Object-Oriented Software**. Addison-Wesley, 1995
- Meszaros, G.; Doble, J. **A pattern language for pattern writing**, cap. 29 in J. Coplien; D. Schmidt. Pattern Languages of Program Design, Reading-MA, Addison-Wesley, p. 529–574, 1998.
- Buschmann F. et. al. **Pattern-oriented software architecture - a system of patterns**. Wiley, 1996.
- Coplien, J. **Advanced C++ programming styles and idioms**. Reading-MA: Addison-Wesley, 1992.
- Shvets, Alexander. **Dive into design patterns**, Refactoring.Guru, 2018, disponível para download em: https://www.goodreads.com/book/show/43125355-dive-into-design-patterns
- Braude, Eric. **Projeto de Software: Da programação à arquitetura: Uma abordagem baseada em Java**, Bookman, 2005.

MBA USP ESALQ

Obrigada!

Profa. Dra. Rosana T. Vaccare Braga
www.linkedin.com/in/rosana-braga-13778912