

**MBA USP
ESALQ**

**PADRÕES DE PROJETO
(DESIGN PATTERNS) I**

Profa. Dra. Rosana T. Vaccare Braga



A responsabilidade pela idoneidade, originalidade
e licitude dos conteúdos didáticos apresentados é do professor.

Proibida a reprodução, total ou parcial, sem autorização.

Lei nº 9610/98

Organização da aula

- Reúso de software e seus benefícios;
- Padrões e sua origem;
 - Patterns vs Standards;
- Elementos de um padrão, tipos de padrão;
- Padrões de projeto: definições, vantagens, categorias, formatos de escrita;
- Padrões GoF da categoria “de criação” e início da categoria “estrutural”.



Jocy Gonçalves Neto 11.989.497-24

Padrões de Software (origem)

- **Reúso de Software:** replicação de qualquer tipo de conhecimento sobre um sistema em outros sistemas similares, com o objetivo de reduzir o esforço de desenvolvimento e a manutenção nesses novos sistemas (Biggerstaff e Perlis, 1989).
 - Anos 70: módulos e sub-rotinas
 - Anos 80: classes e geradores de aplicação
 - Anos 90: análise de domínio, componentes, padrões (*patterns*) e frameworks;
 - Anos 2000: serviços, aspectos
 - ...
- Benefícios da reutilização:
 - Melhores índices de produtividade
 - Produtos de melhor qualidade, mais confiáveis, consistentes e padronizados
 - Redução dos custos e tempo envolvidos no desenvolvimento de software
 - Maior flexibilidade na estrutura do software produzido, facilitando sua manutenção e evolução

Importante:
patterns x standards

Importante: Patterns x Standards

Tema deste módulo
do MBA

Patterns:

- Proxy, Composite, Factory-method, etc (**de projeto**)
- Camadas, MVC, etc (arquiteturais)
- Transaction, Item-transaction, Type-object (de análise)
- Site map, shopping cart, ... (de interação com o usuário)
- etc...

Standards:

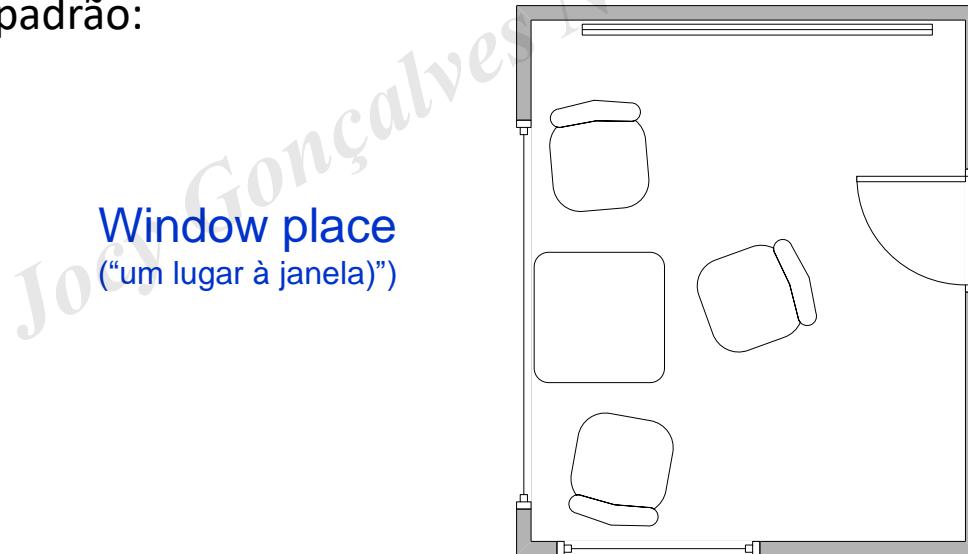
- USB, HDMI, Bluetooth, ...
- HTML, XML, ...
- ISO 9000, ISO/IEC 15504, ...
- IEEE
- etc...

Patterns: [The Hillside Group - A group dedicated to design patterns. Home of the patterns library.](http://www.hillsidegroup.com/)

Padrões de Software (origem)

- Origem dos padrões:

- Christopher Alexander (1977, 1979) - Arquiteto
 - “Cada padrão descreve um problema que ocorre repetidas vezes em nosso ambiente, e então descreve o núcleo da solução para esse problema, de forma que você possa utilizar essa solução milhões de vezes sem usá-la do mesmo modo duas vezes”
 - Proposta de padrões extraídos a partir de estruturas de edifícios e cidades de diversas culturas, com o intuito de ajudar as pessoas a construir suas comunidades com a melhor qualidade de vida possível
 - Exemplo de um padrão:

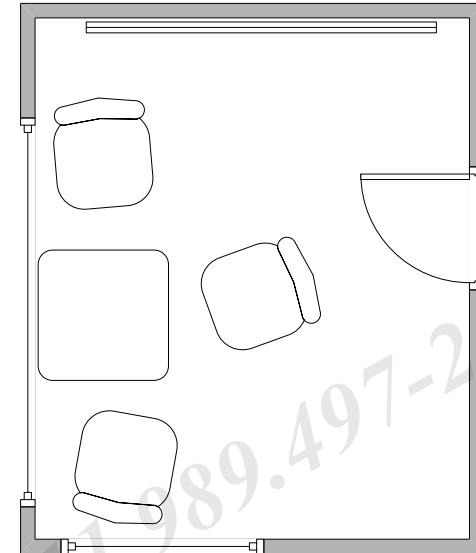


Window place
("um lugar à janela")

*Christopher Alexander et al, *A Pattern Language-Towns. Buildings. Construction*, 1977

Padrões de Software (origem)

Window place
("um lugar à janela")



*Christopher Alexander et al, *A Pattern Language- Towns.Buildings. Construction*, 1977

- Todo mundo adora assentos na janela, janelas salientes e grandes janelas com peitoris baixos e cadeiras confortáveis puxadas para eles ... Uma sala sem um lugar como este raramente permite que você se sinta confortável
- Se a sala não contiver janela que seja um "lugar", uma pessoa será levada a :
 - 1. Querer sentar e ficar confortável OU
 - 2. Ser atraído para a luz
- Se lugares confortáveis estão longe das janelas, não há como superar esse conflito...
- Portanto: em todos os cômodos onde você passa algum tempo durante o dia, faça pelo menos um "lugar à janela"

Padrões de Software (origem)

Beck e Cunningham: 1987

pequena linguagem de padrões para guiar programadores inexperientes em Smalltalk

Peter Coad: 1992

padrões de análise descobertos na modelagem de sistemas de informação

James Coplien – 1992

catálogo de vários estilos (“idioms”), com o intuito de padronizar a escrita de código em C+

Gamma et al – 1995

padrões de projeto derivados a partir de experiência prática com desenvolvimento de software orientado a objetos

Em software:

1977	1979	1987	1992	1995
<i>A Pattern Language</i> Alexander C.	<i>A timeless way of building</i> Alexander C. Livros de arquitetura que deram a base para os padrões de software	<i>Using Pattern Languages for Object-Oriented Programs</i> Beck e Cunningham Padrões de projeto para linguagem Smalltalk	<i>Object-oriented patterns</i> Peter Coad <i>Advanced C++ programming styles and idioms</i> James O. Coplien	<i>Design Patterns Elements of Reusable Object-Oriented Software</i> Gamma et al. Definição dos primeiros de seus 23 padrões existentes hoje.

Por que Padrões ?

- Desenvolvedores acumulam soluções para os problemas que resolvem com frequência
- Essas soluções são difíceis de serem elaboradas e podem aumentar a produtividade, qualidade e uniformidade do software
- Como documentar essas soluções de forma que outros desenvolvedores, menos experientes, possam utilizá-las?

Padrões de Software: Descrevem soluções para problemas que ocorrem com frequência no desenvolvimento de software (*Gamma 95*)

- **Um bom padrão:**
 - resolve um problema
 - é um conceito aprovado (não apenas teoria)
 - a solução não é óbvia
 - descreve um relacionamento (estruturas e mecanismos)
 - tem um componente humano significativo

Vantagens de Padrões

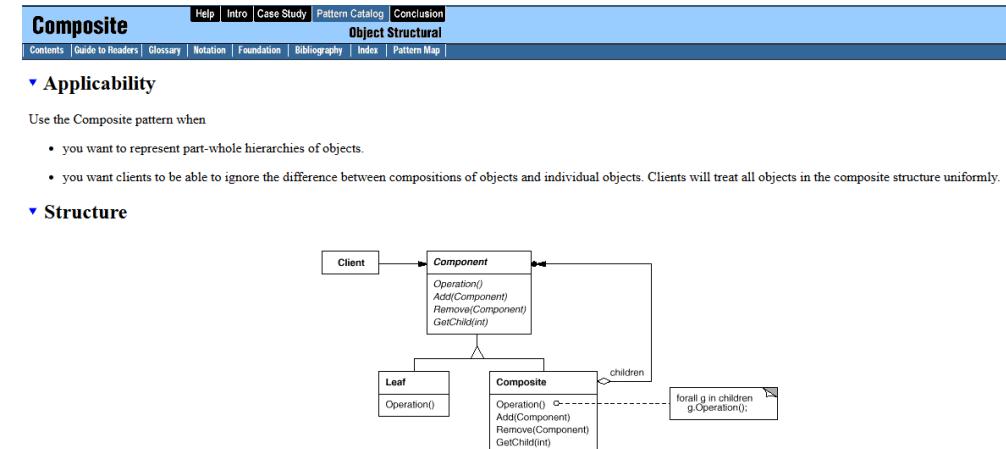
- **Aumento de produtividade:** aplicação de soluções prontas leva a menos tempo em pensar na solução, projetá-la para depois aplicá-la.
- **Uniformidade na estrutura do software:** entender um software fica mais fácil quando padrões foram utilizados, pois consegue-se enxergar os padrões utilizados ao longo do software (seja no projeto ou na implementação).
- **Aplicação imediata por outros desenvolvedores:** conhecendo o padrão (seu propósito, sua estrutura, o contexto em que deve ser aplicado, suas consequências, etc.) pode-se aplicá-lo imediatamente mesmo sem ter sido o proponente do padrão.
- **Redução da complexidade:** padrões passam a ser vistos como blocos construtivos, em que a complexidade inerente a cada padrão é abstraída e vê-se o todo com menor complexidade (princípio dividir para conquistar).

Categorias de Padrões

- Organizacionais
- Arquiteturais
- De Análise
- De Projeto
 - **Gamma et al**
 - Outros : p.ex J2EE
- De programação (estilos)
- De usabilidade
- Educacionais
- Etc.

Tema deste módulo
do MBA

Exemplo



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

Elementos de um padrão

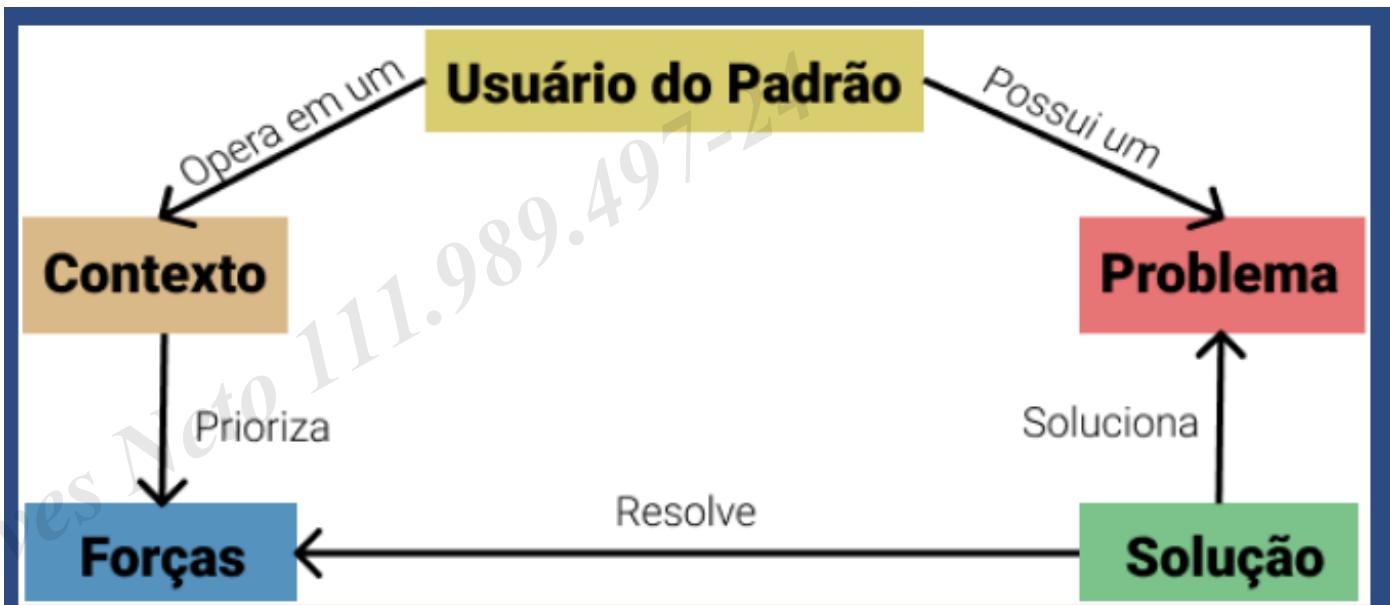


Elementos de um padrão



Enquete 1 : Problema:

“não tenho dinheiro para fazer minha viagem de férias”



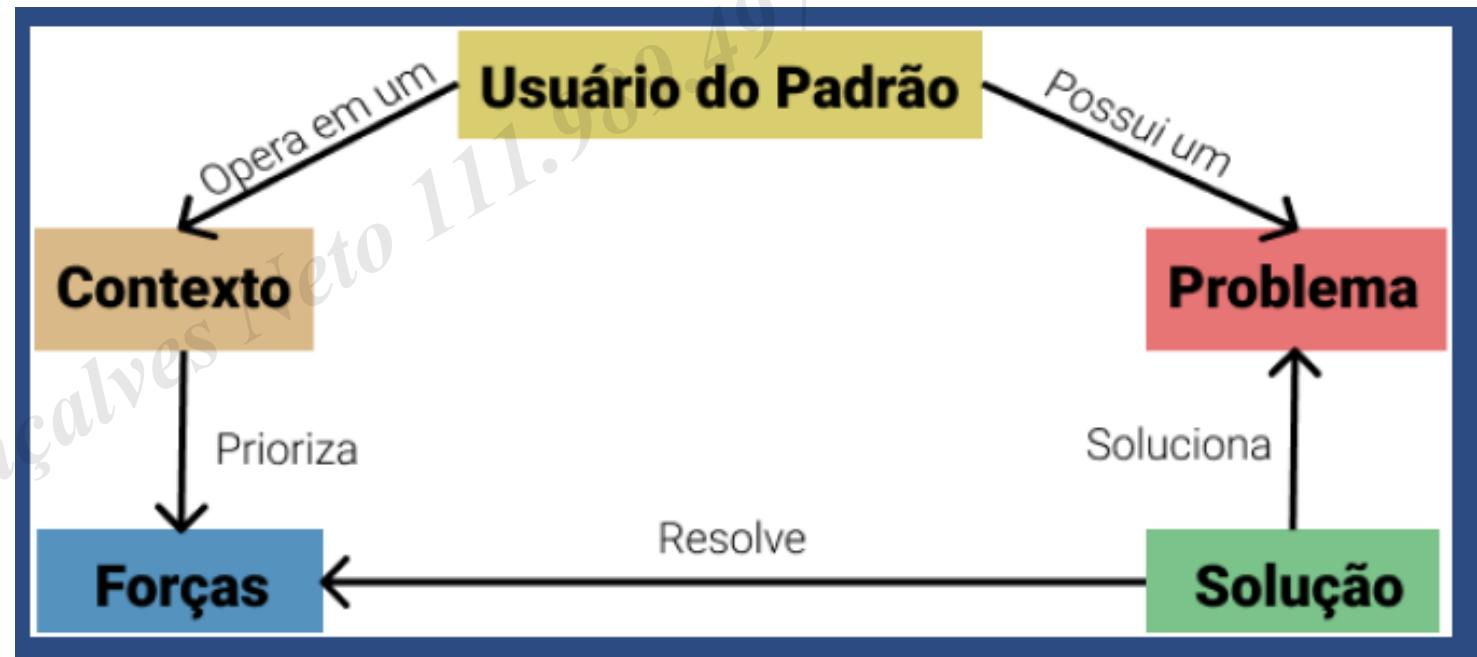
(Meszaros & Doble, 1998)

Elementos de um padrão



Enquete 2 : Problema num dado contexto:

“não tenho dinheiro para fazer minha viagem de férias”



(Meszaros & Doble, 1998)

Formato para escrita de padrões de software

- Ao escrever um padrão, pode-se combinar alguns elementos essenciais e outros elementos opcionais.
- A princípio, não deve se prender a nenhum formato de escrita e não há obrigatoriedade de uso de todos os elementos em todos os padrões.
- Na medida em que o padrão vai sendo reusado, pode-se mudar seu formato para facilitar seu entendimento e reuso
- Existem algumas sugestões de gabaritos já testados e aprovados : **Gamma et al. (1995)**, **Coplien (1996)** e **Buschmann et al. (1996)**

Formato para escrita de padrões de software

Tema deste
módulo do MBA

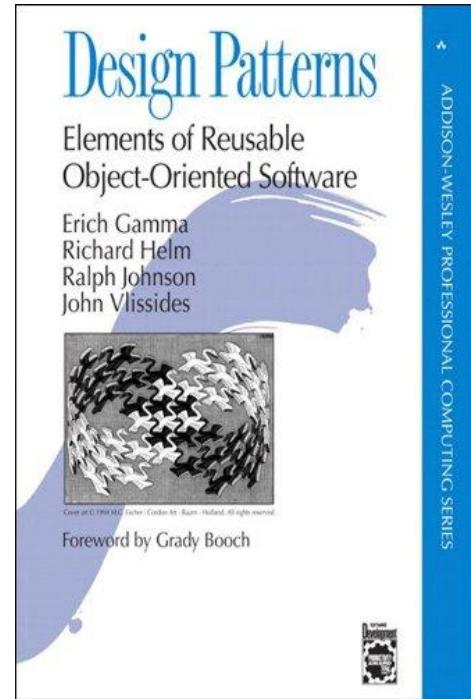
Gamma	Coplien	Buschmann
Nome	Nome	Nome
Classificação	Contexto	Também Conhecido Como
Intenção	Problema	Exemplo
Também Conhecido Como	Forças	Contexto
Motivação	Solução	Problema
Aplicabilidade	Esquema	Solução
Estrutura	Contexto	Estrutura
Participantes	Resultante	Dinâmica
Colaborações	Raciocínio	Implementação
Implementação		Exemplo Resolvido
Código Exemplo		Variantes
Consequências		Usos Conhecidos
Usos Conhecidos		Consequências
Padrões Relacionados		Veja Também

PADRÕES DE PROJETO:

**Descrevem soluções para problemas que ocorrem com frequência
no projeto de software orientado a objetos**

Tipos de Padrões de Projeto GoF*

- Catálogo de Padrões de Projeto publicado em 1995 por Gamma, Helm, Johnson e Vlissides
- Dois critérios de classificação:
 - Propósito - reflete o que o padrão faz
 - De Criação: trata da criação de objetos
 - Estrutural: cuida da composição de classes e objetos
 - Comportamental: caracteriza o modo como as classes e objetos interagem e distribuem responsabilidades
 - Escopo em que o padrão atua:
 - Classe: trata do relacionamento entre classes e subclasses (herança - relacionamento estático)
 - Objetos: lida com a manipulação de objetos (podem ser modificados em tempo de execução)



* GoF: Gang of Four – apelido dado aos quatro autores do livro

Padrões de Projeto GoF



Enquete 3: Experiência
com padrões GoF



		Propósito		
		De criação	Estrutural	Comportamental
Classe	Método-fábrica	Adaptador	Interpretador Método gabarito	
Escopo	Objeto	<ul style="list-style-type: none">• Fábrica Abstrata• Construtor• Protótipo• Objeto Unitário (Singleton)	<ul style="list-style-type: none">• Adaptador• Ponte• Composto• Decorador• Fachada• Peso-pena• Procurador	<ul style="list-style-type: none">• Cadeia de Responsabilidade• Comando• Iterador• Mediador• Memento• Observador• Estado• Estratégia• Visitador

Antes de começar...

- Vamos entender como a GoF estruturou a escrita dos 23 padrões de projeto?
 - **Nome e classificação do padrão:** O nome do padrão transmite a essência do padrão sucintamente. Um bom nome é vital, porque ele se tornará parte do seu vocabulário de projeto. A classificação do padrão reflete o esquema apresentado no slide anterior
 - **Intenção:** Uma breve declaração que responde às seguintes perguntas: O que o padrão de projeto faz? Qual é sua justificativa e intenção? Que problema ou questão de design em particular ele aborda?
 - **Também conhecido como:** Outros nomes conhecidos para o padrão, se houver.
 - **Motivação:** Um cenário que ilustra um problema de projeto e como as estruturas de classes e objetos no padrão resolvem o problema. O cenário ajudará você a entender a descrição mais abstrata do padrão que se segue.

Antes de começar...

- **Aplicabilidade:** Quais são as situações em que o padrão de projeto pode ser aplicado? Quais são os exemplos de projetos ruins que o padrão pode abordar? Como você pode reconhecer essas situações?
- **Estrutura:** No caso do livro da GoF, foi adotada uma representação gráfica das classes no padrão usando uma notação baseada na Técnica de Modelagem de Objetos (*Object Modeling Technique*, uma precursora da UML (*Unified Modeling Language*)). Também são usados diagramas de interação [JCJO92, Boo94] para ilustrar sequências de solicitações e colaborações entre objetos.
- **Participantes:** As classes e/ou objetos que participam do padrão de projeto e suas responsabilidades.
- **Colaborações:** Como os participantes colaboram para executar suas responsabilidades.

Antes de começar...

- **Consequências:** Como o padrão suporta seus objetivos? Quais são as compensações e os resultados do uso do padrão? Qual aspecto da estrutura do sistema ele permite que você varie independentemente?
- **Implementação:** Quais armadilhas, dicas ou técnicas você deve estar ciente ao implementar o padrão? Existem problemas específicos da linguagem?
- **Código de exemplo:** Fragmentos de código que ilustram como você pode implementar o padrão em uma linguagem OO.
- **Usos conhecidos:** Exemplos do padrão encontrados em sistemas reais. O livro inclui pelo menos dois exemplos de domínios diferentes.
- **Padrões relacionados:** Quais padrões de projeto estão intimamente relacionados a este? Quais são as diferenças importantes? Com quais outros padrões este deve ser usado?

Escopo	Propósito		
	De criação	Estrutural	Comportamental
	Classe	Método-fábrica	Adaptador
Objeto	<ul style="list-style-type: none"> • Fábrica Abstrata • Construtor • Protótipo • Objeto Unitário (Singleton) 	<ul style="list-style-type: none"> • Adaptador • Ponte • Composto • Decorador • Fachada • Peso-pena • Procurador 	<ul style="list-style-type: none"> • Cadeia de Responsabilidade • Comando • Iterador • Mediador • Memento • Observador • Estado • Estratégia • Visitador

PADRÕES da categoria “criacionais” ou “de criação”

Encarregados do processo de criação de objetos, podendo aumentar a flexibilidade e reutilização de código

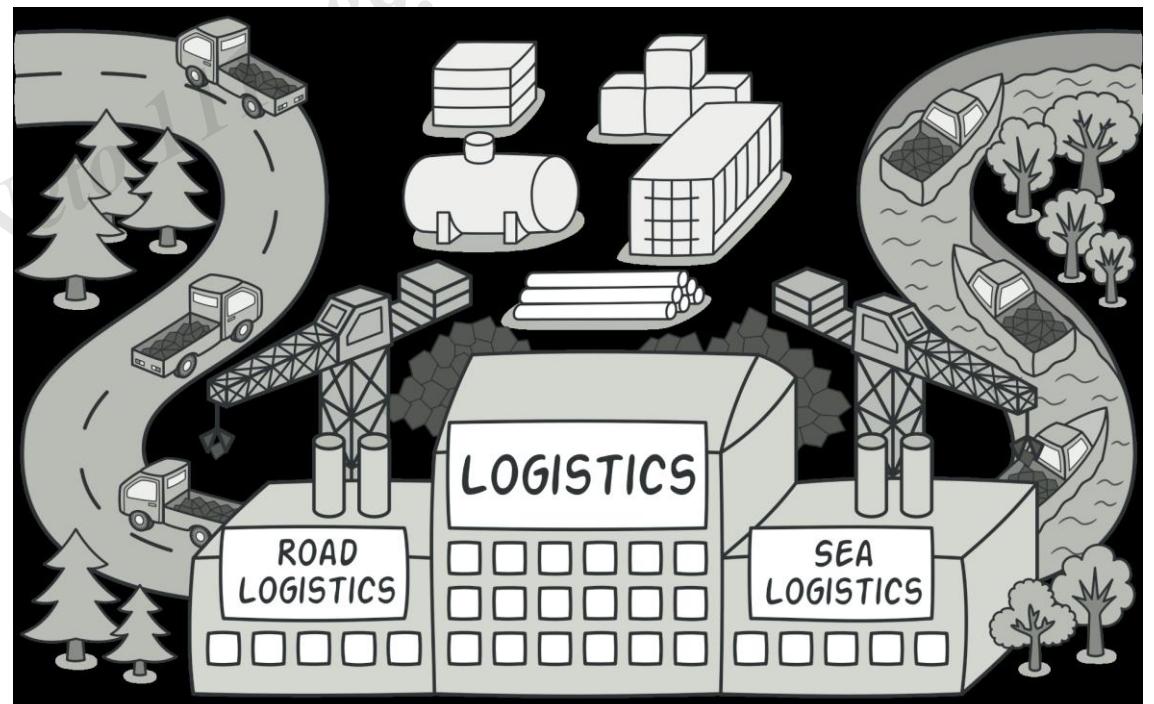
Padrão de Projeto Método-fábrica (do inglês Factory Method)

```
Product p1,p2;  
Truck t1 = new Truck();  
t1.loadTruckWithProduct(p1);  
p1.deliverByTruck(t1);  
Ship s1 = new Ship();  
s1.loadShipWithProduct(p2);  
p2.deliverByShip(s1);
```

O que esse
código implica?



Enquete 4 :
Truck, Ship...



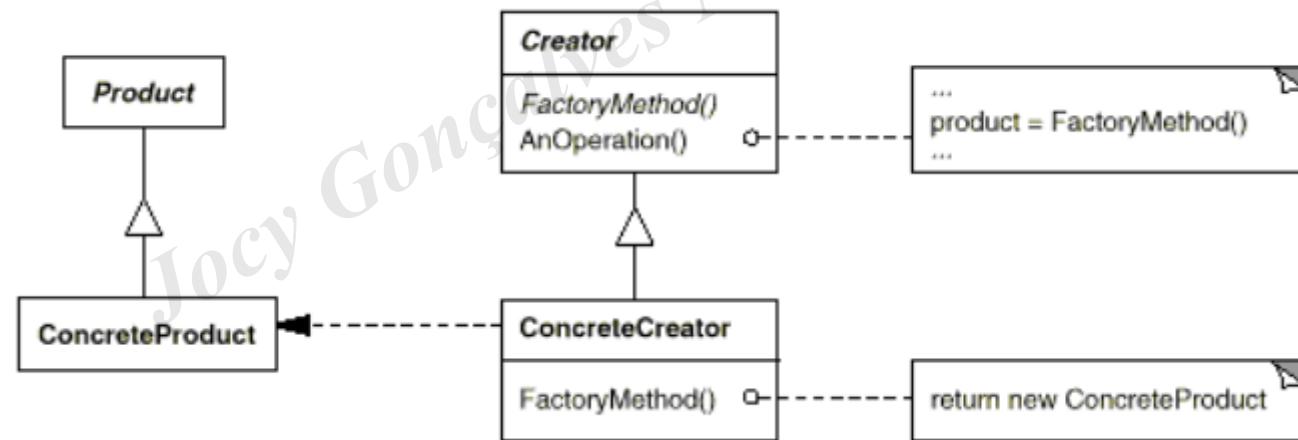
Shvets, Alexander, 2019

Padrão de Projeto Método-fábrica (do inglês Factory Method)

- **Classificação:** Padrão Estrutural/Escopo de classes
- **Intenção:** Definir uma interface para criar um objeto, mas deixar que as subclasses decidam qual classe instanciar. O padrão permite que uma classe adie a instanciação para subclasses.
- **Motivação:** Em vários tipos de aplicações, a escolha de qual objeto será criado depende de escolhas feitas em tempo de execução, portanto não há como saber que classe deve ser instanciada. Isso pode causar muito código duplicado para prever as diversas combinações de classes, além de estruturas case espalhadas pelo código.

Padrão de Projeto Método-fábrica

- Aplicabilidade
- Use o padrão Método-fábrica quando:
 - uma classe não pode antecipar a classe de objetos que deve criar.
 - uma classe quer que suas subclasses especifiquem os objetos criados por elas.
- Estrutura:



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

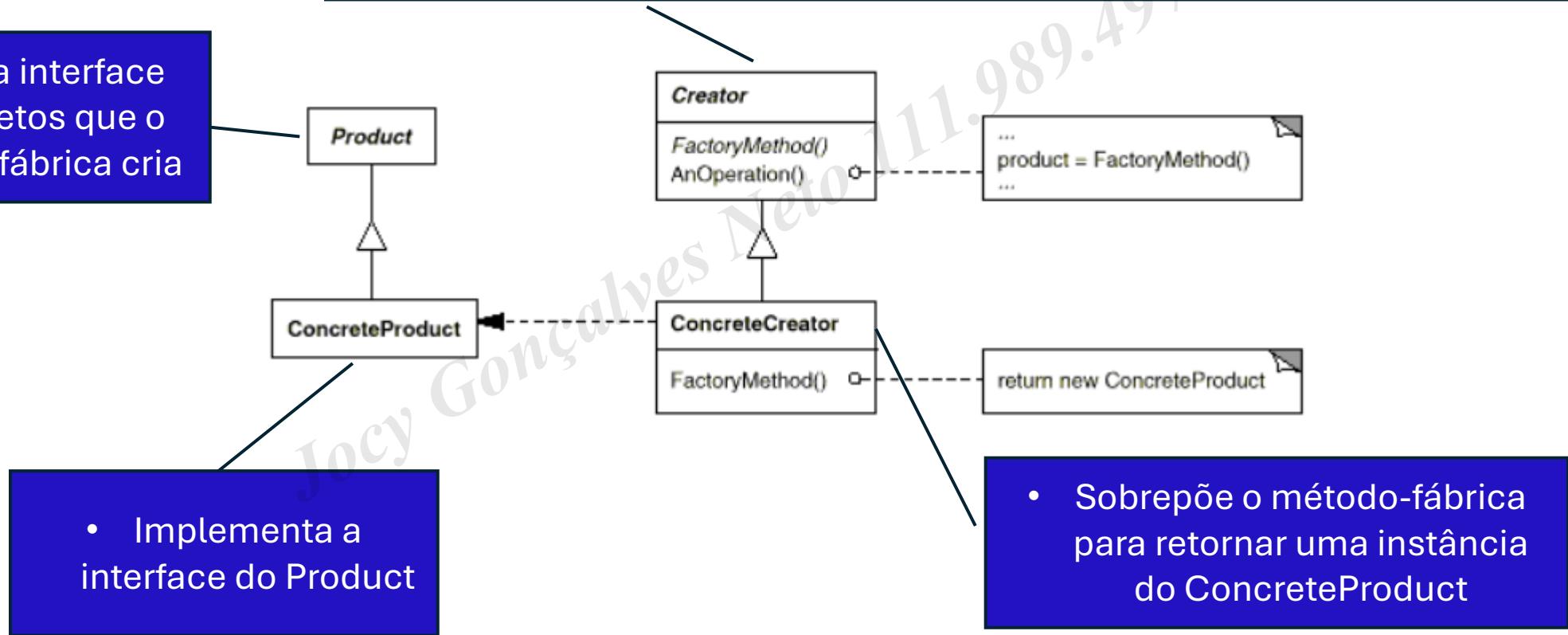
Padrão de Projeto Método-fábrica

- **Participantes:**

- Define a interface dos objetos que o método-fábrica cria

- Implementa a interface do Product

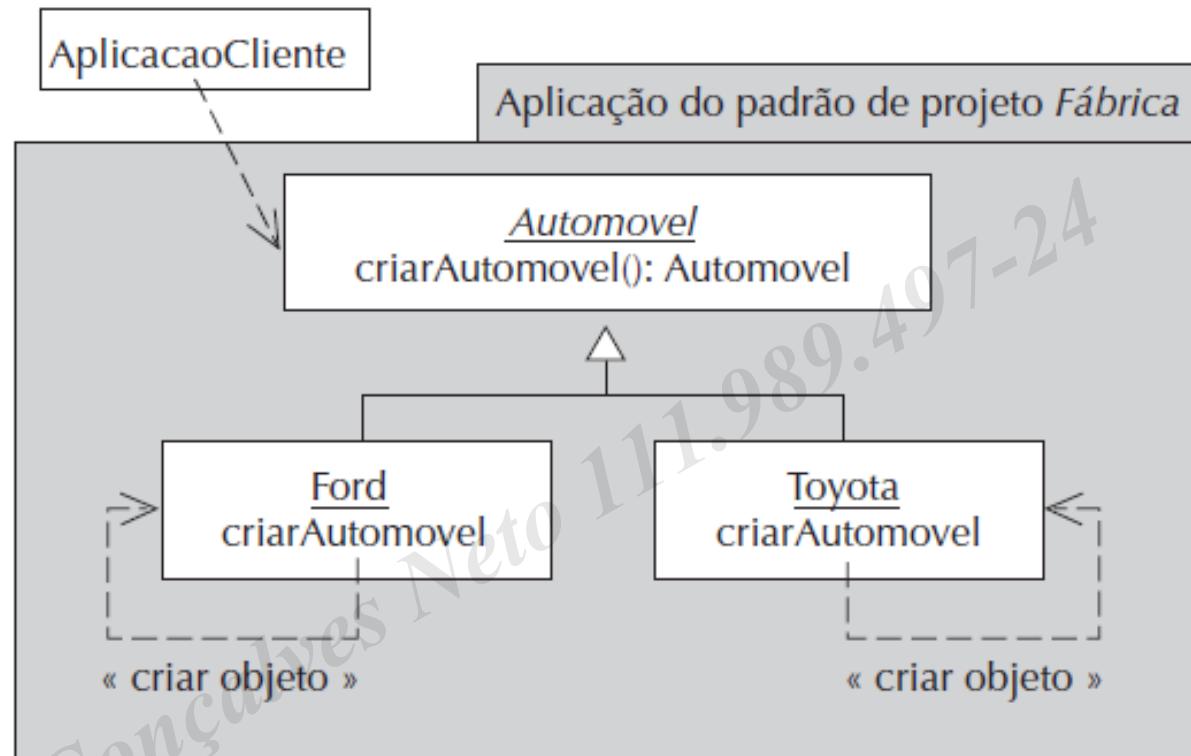
- Declara o método-fábrica, que retorna um objeto do tipo Product. Pode também definir uma implementação default que retorna um objeto ConcreteProduct.
 - Pode chamar o método-fábrica para criar um objeto Product



- Sobrepõe o método-fábrica para retornar uma instância do ConcreteProduct

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

Exemplo de código do Padrão de Projeto Método-Fábrica



Braude, Eric, 2005

Exemplo de código do Padrão de Projeto Método-Fábrica

```
abstract class Automovel // essa é a classe Product
{
    int anoFab;
    int getAnoFab()
    {
        return this.anoFab;
    }
    . . .           // Outros atributos de automóvel entram aqui
    public double calcularSeguro(int anoFab)
    {
        return 0.00; // comportamento default da classe
    }
}

class Ford extends Automovel // essa é a classe ConcreteProduct
{ . . .
    double calcularSeguro()
    {
        return this.anoFab + this.anoFab*3.123/2.12; // regras específicas da Ford
    }
}
.
.
.
class Toyota extends Automovel // essa é mais uma classe ConcreteProduct
{ . . .
    double calcularSeguro()
    {
        return this.anoFab*2 - this.anoFab*4.321/2.35; // regras específicas da Toyota
    }
}
.
```

Exemplo de código do Padrão de Projeto Método-Fábrica

```
abstract class Corretora // essa é a classe Creator
{
    public Automovel criarAutomovel() // método-fábrica
    {
        }
    void renovarSeguro() // anOperation
    {
        Automovel aut = criarAutomovel();
        int ano = aut.obterAnoFab();
        double v = aut.calcularSeguro();
        System.out.println( "Seguro renovado!!" );
    }
}

class CriadorFord extends Criador // essa é a classe ConcreteCreator
{
    Ford criarAutomovel()
    {
        return new Ford();
    }
}
class CriadorToyota extends Criador // outra classe ConcreteCreator
{
    Toyota criarAutomovel()
    {
        return new Toyota();
    }
}
```

Exemplo de código do Padrão de Projeto Método-Fábrica

```
public static void main( String[] args )
{
    Automovel meuAuto = obterTipoAutomovel();
    Corretora cor = new Corretora;
    cor.renovarSeguro(meuAuto); }
}

private static Automovel obterTipoAutomovel()
{
    System.out.println( "Por favor, escolha 1 para Ford ou 2 para Toyota");
    try // seleciona a entrada do usuário
    { BufferedReader bufReader =
        new BufferedReader( new InputStreamReader( System.in ) );
        tipoDesejado = bufReader.readLine();
    }
    catch( IOException e )
    { System.out.println( e );
    }
    // Retorna a instância de 'Automovel' de acordo com a entrada do usuário
    // este é o trecho que precisa ser alterado se quiser disponibilizar mais
    tipos de automóvel
    if( tipoDesejado == 1)
        autoSelecionado = CriadorFord.criarAutomovel();
    else
        autoSelecionado = CriadorToyota.criarAutomovel();
    return autoSelecionado;
}
```

Consequências de uso do Método-Fábrica

- Evita acoplamento forte entre o criador e os produtos concretos
- Fica mais fácil incluir novos tipos de produtos concretos, sem quebrar o código
- **Ponto negativo:**
 - O código fica mais complexo já que é necessário incluir novas classes para implementar o padrão

Padrões relacionados ao Método-Fábrica

- Abstract Factory (veremos em seguida!)
- Template Method
- Prototype



**Enquete 5 : Uso do
método-fábrica**

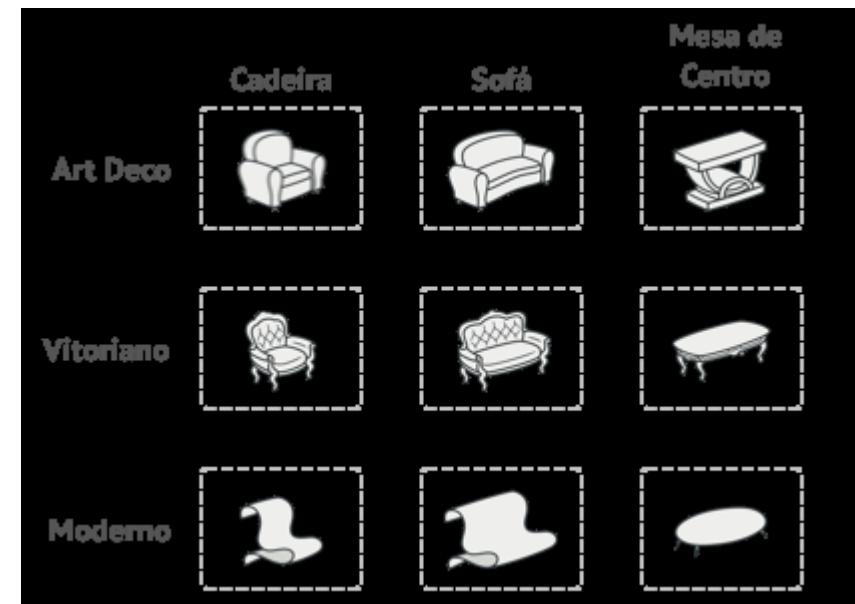


Padrão de Projeto Fábrica abstrata (do inglês Abstract Factory)



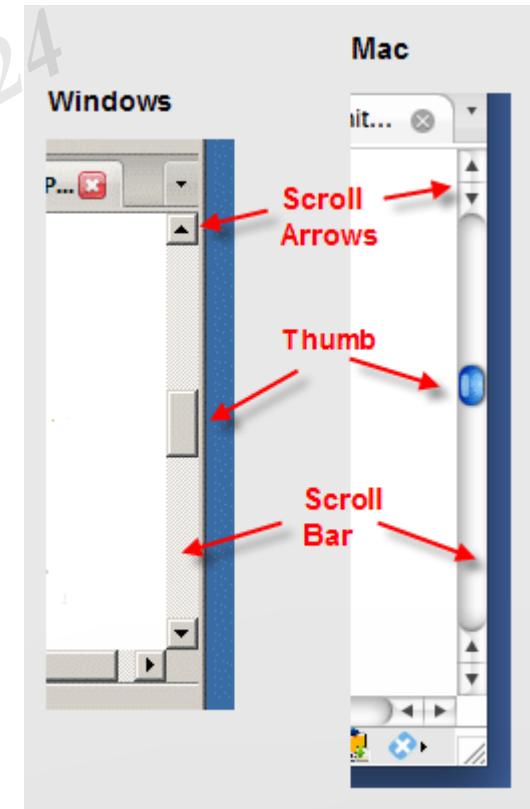
Shvets, Alexander, 2019

E se eu precisar
fabricar famílias de
objetos relacionados
?



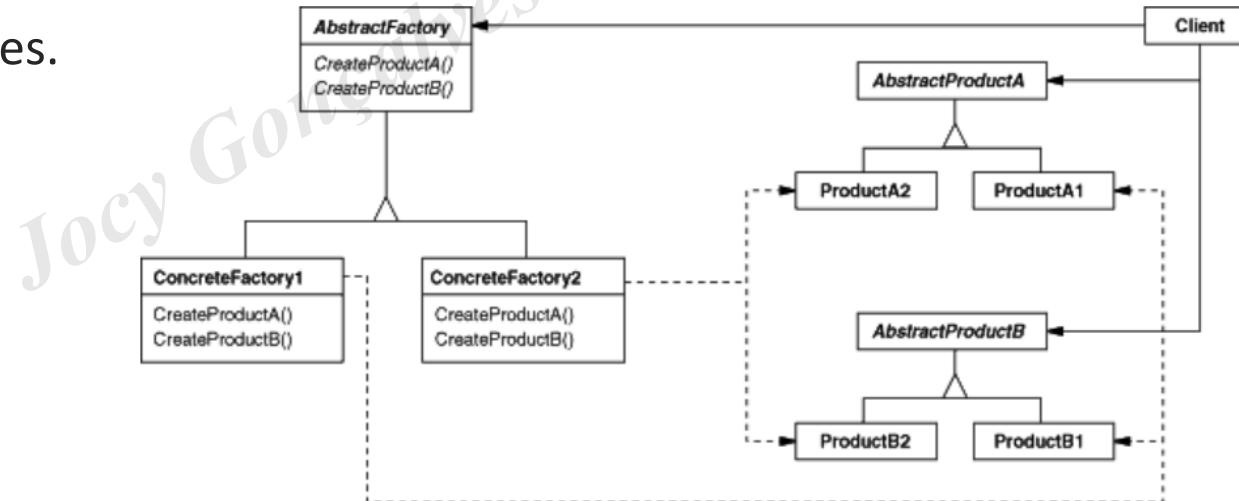
Padrão de Projeto Fábrica abstrata

- **Classificação:** Padrão de criação/Escopo de objetos
- **Intenção:** Fornecer uma interface para criar famílias de objetos relacionados ou dependentes sem especificar suas classes concretas.
- **Motivação:** Em muitas aplicações, é necessário criar um conjunto de objetos relacionados, mas que podem ter implementação diferente em cada plataforma.
 - Por exemplo, num sistema de interface gráfica pode haver “widgets” como botões, janelas, barras de rolagem para Windows ou para MacOS. Se for necessário fazer a portabilidade entre diferentes S.O.s, o código estará amarrado a certas classes e precisa ser inteiramente modificado
 - Escolhido um estilo, todos os elementos devem ser sempre utilizados em conjunto
 - Mas há comportamentos comuns entre os elementos das diferentes famílias, por exemplo: um botão sempre tem os mesmos tipos de ação: clique (esquerdo e direito), rolamento, etc., seja Windows ou MacOS.



Padrão de Projeto Fábrica abstrata

- **Aplicabilidade** - Use o padrão Fábrica abstrata quando:
 - um sistema deve ser independente de como seus produtos são criados, compostos e representados.
 - um sistema deve ser configurado com uma das várias famílias de produtos.
 - uma família de objetos de produtos relacionados é projetada para ser usada em conjunto, e você precisa impor essa restrição.
 - você quer fornecer uma biblioteca de classes de produtos, e quer revelar apenas suas interfaces, não suas implementações.
- **Estrutura:**

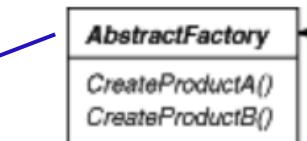


** Imagem capturada no livro versão html
publicado em CD (Gamma et al 1995)

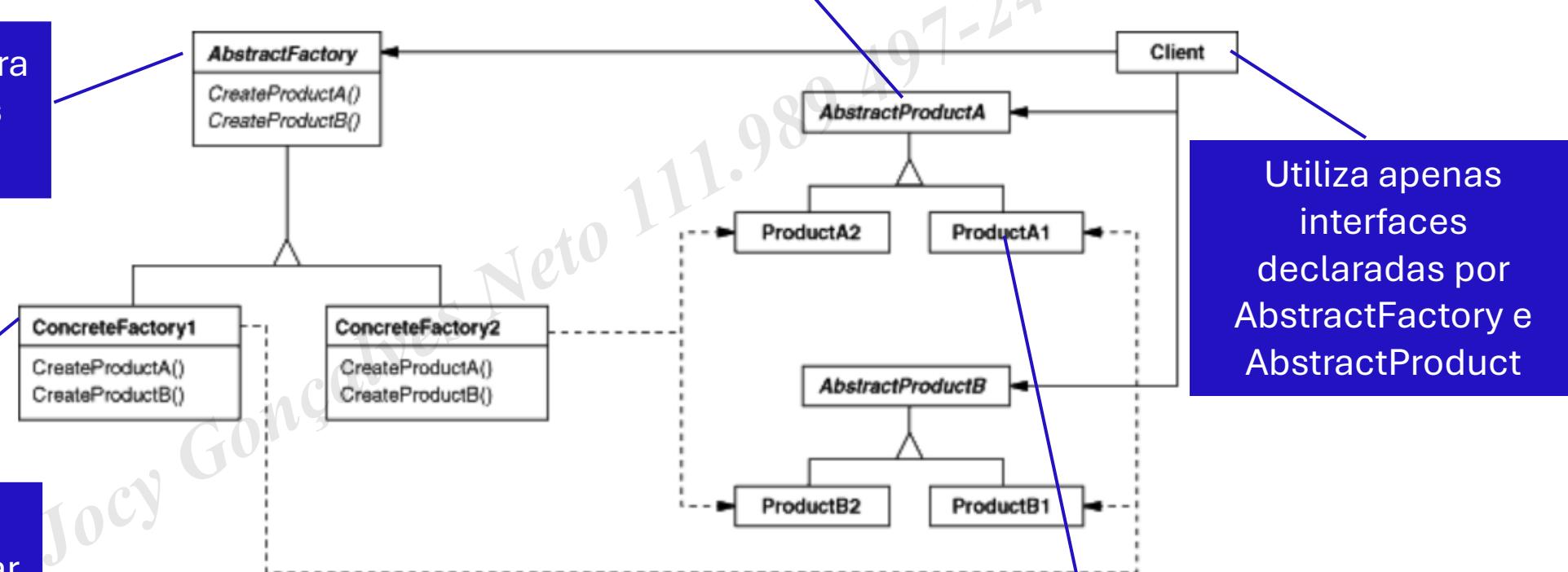
Padrão de Projeto Fábrica abstrata

- Participantes:

- Declara uma interface para operações que criam os objetos abstratos



- Implementa as operações para criar os objetos concretos



- Declara uma interface para um tipo de produto

Utiliza apenas interfaces declaradas por **AbstractFactory** e **AbstractProduct**

- Define um objeto a ser criado pela fábrica concreta correspondente
 - Implementa a interface **AbstractProduct**

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

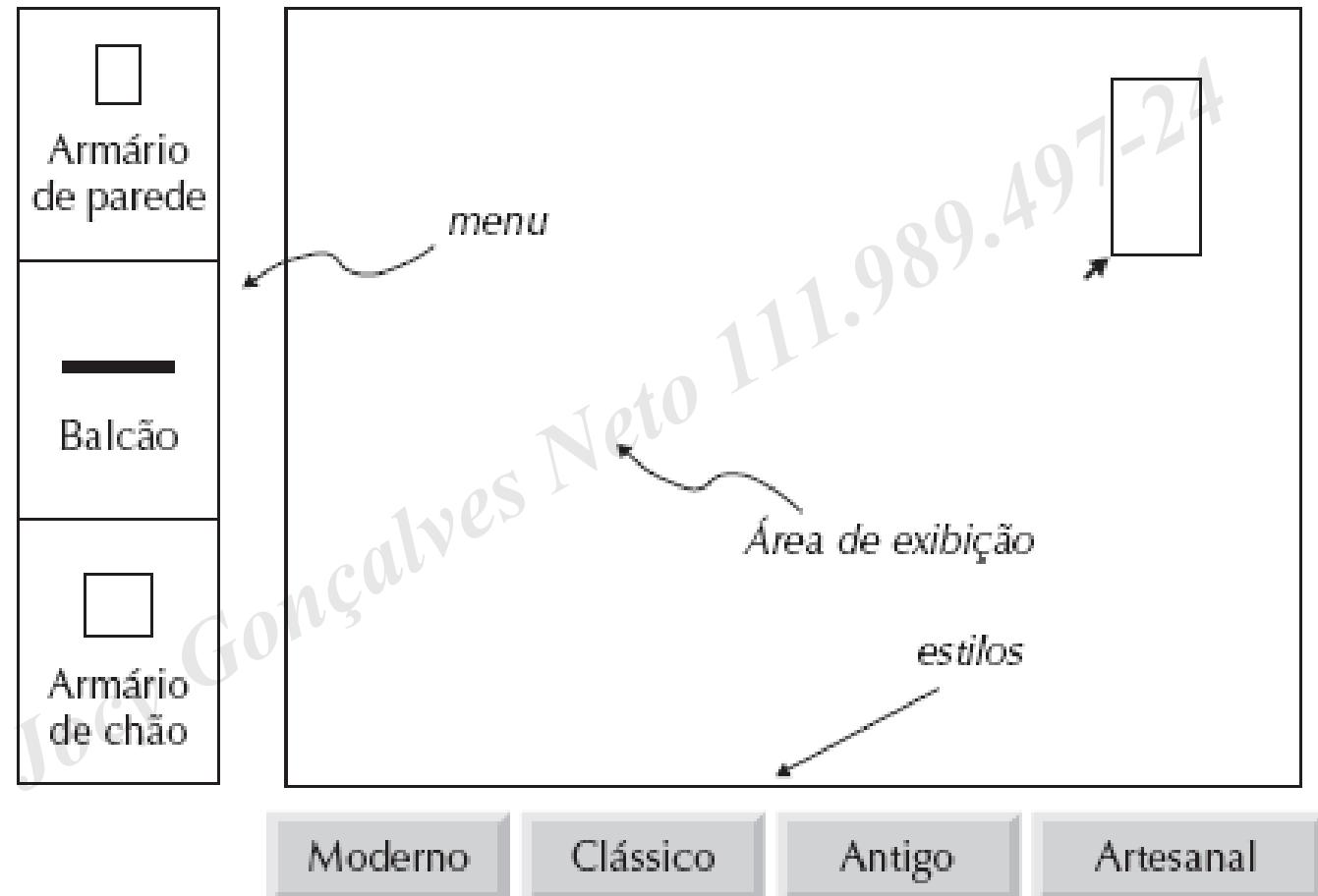
Padrão de Projeto Fábrica abstrata

Exemplo do Padrão Fábrica abstrata

- **Aplicação para construir armários de cozinha**
- Proprietários de residências sempre têm a intenção de modernizar suas cozinhas, frequentemente utilizando um software para visualizar as possibilidades.
- Visualizador De Cozinhas: aplicação que permite que o usuário crie o layout das partes de uma cozinha, sem comprometer-se com um estilo.

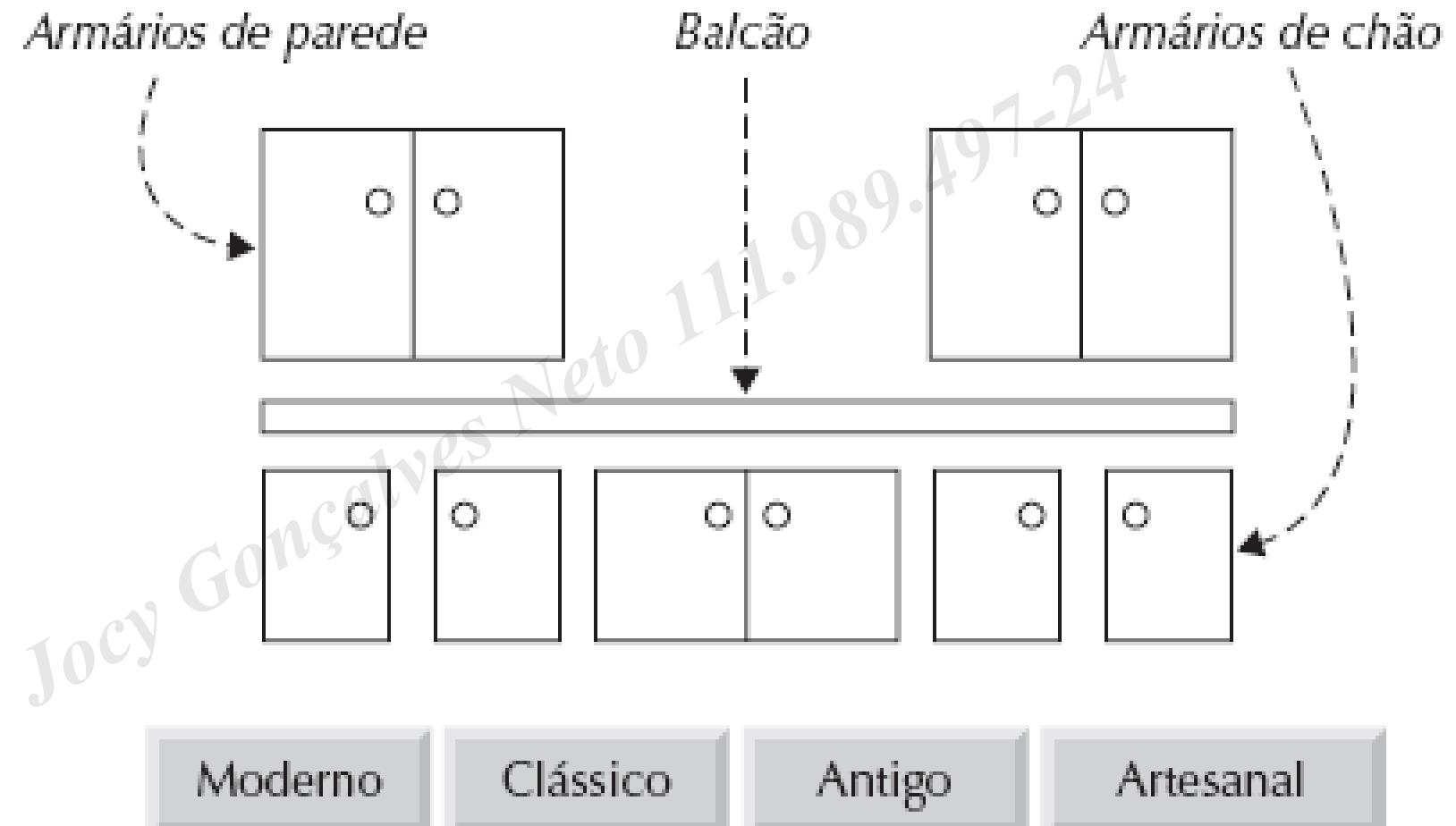
Padrão de Projeto Fábrica abstrata

Exemplo do
Padrão
Fábrica
abstrata



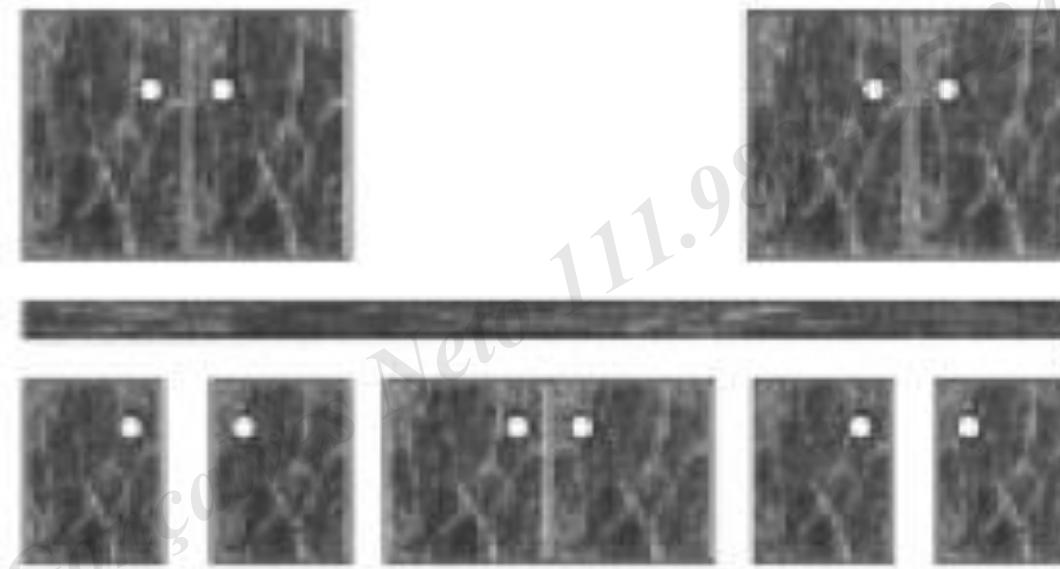
Padrão de Projeto Fábrica abstrata

Exemplo do
Padrão
Fábrica
abstrata



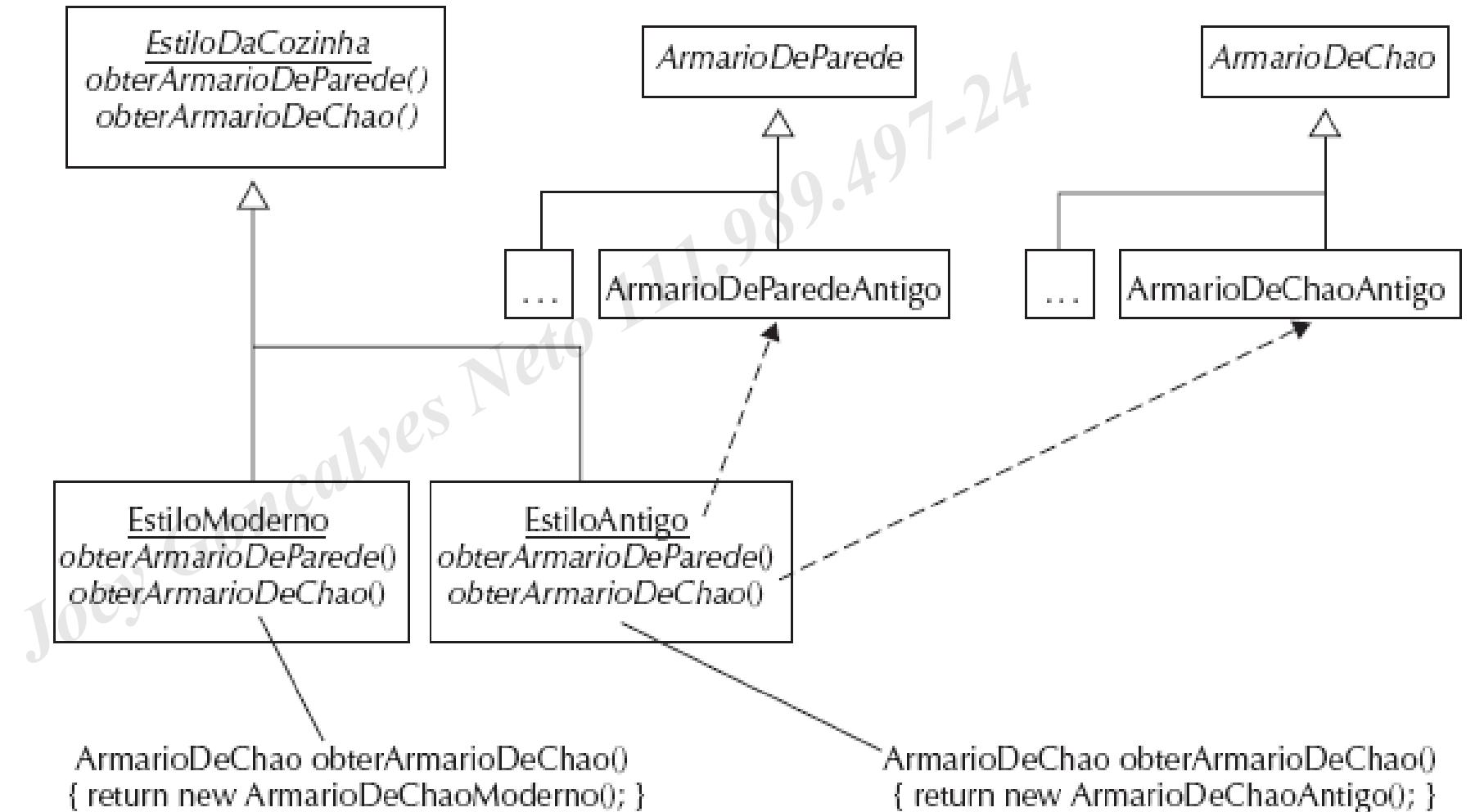
Padrão de Projeto Fábrica abstrata

Exemplo do
Padrão
Fábrica
abstrata



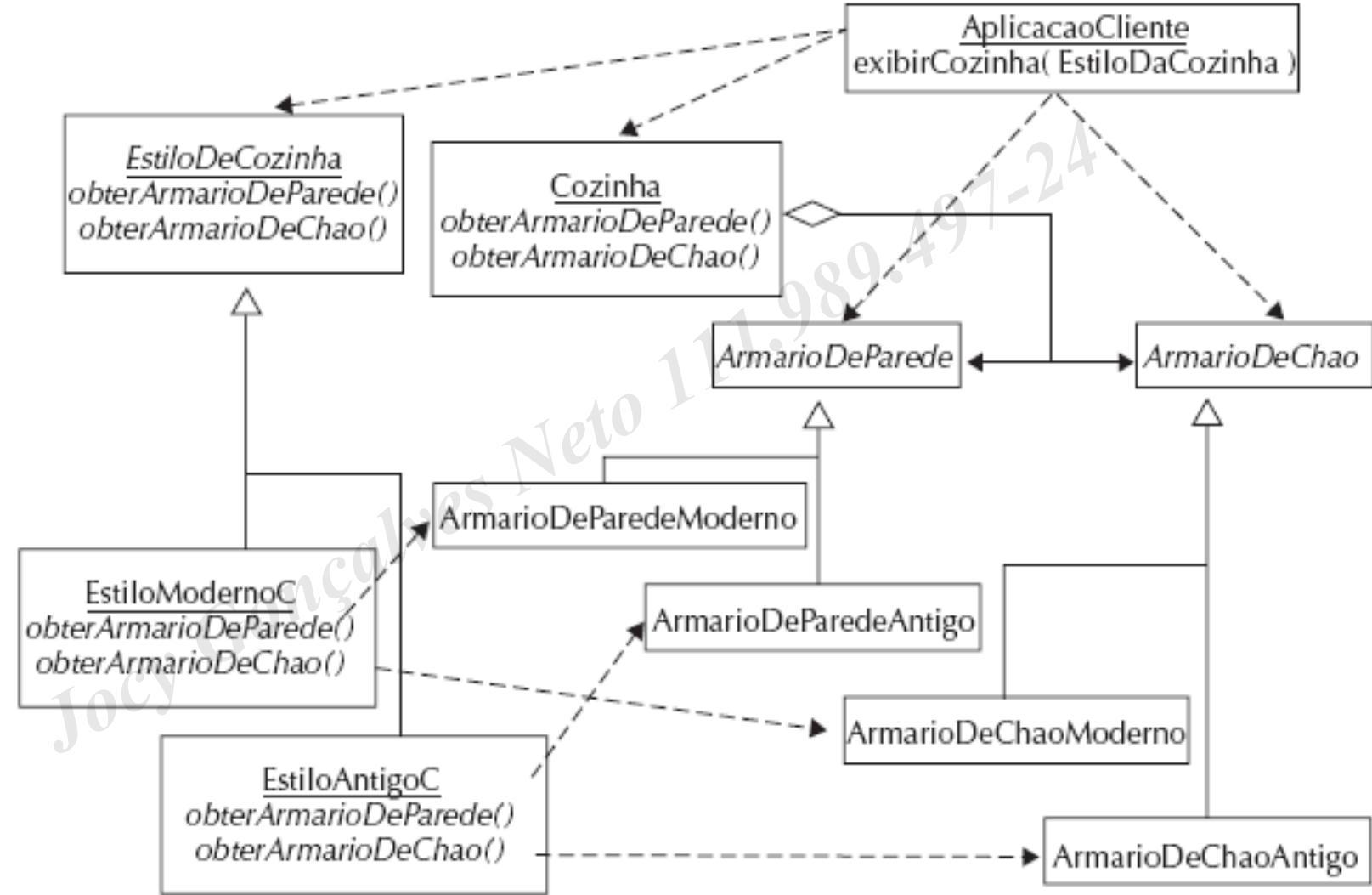
Padrão de Projeto Fábrica abstrata

Exemplo do
Padrão
Fábrica
abstrata



Padrão de Projeto Fábrica abstrata

Exemplo do
Padrão
Fábrica
abstrata



Padrão de Projeto Fábrica abstrata

Exemplo do Padrão Fábrica abstrata

```
//Determina o estilo instanciando meuEstilo  
EstiloAntigoC meuEstilo = new EstiloAntigoC;  
// Cria os armários de parede: Tipo determinado pela classe de  
meuEstilo  
ArmarioDeParede ArmarioDeParede1 = meuEstilo.obterArmarioDeParede();  
ArmarioDeParede ArmarioDeParede2 = meuEstilo.obterArmarioDeParede();  
. . .  
// Cria os armários de chão: Tipo determinado pela classe de  
meuEstilo  
// Cria o objeto cozinha (no estilo requerido)  
ArmarioDeChao armarioDeChao1 = meuEstilo.obterArmarioDeChao();  
ArmarioDeChao armarioDeChao2 = meuEstilo.obterArmarioDeChao();  
. . .  
Cozinha cozinha = new Cozinha();  
Cozinha.adicionar( armarioDeParede1, . . . );  
Cozinha.adicionar( armarioDeParede2, . . . );  
. . .  
Cozinha.adicionar( armarioDeChao1 . . . );  
Cozinha.adicionar( armarioDeChao2 . . . );  
. . .
```

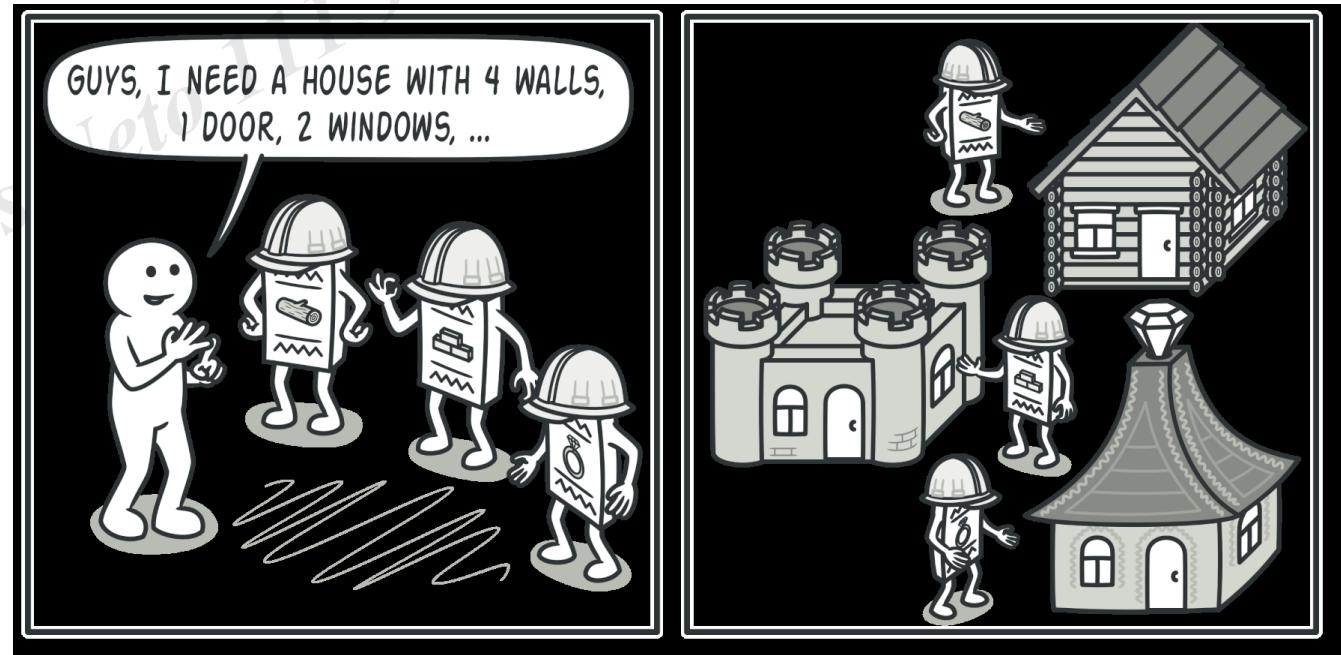
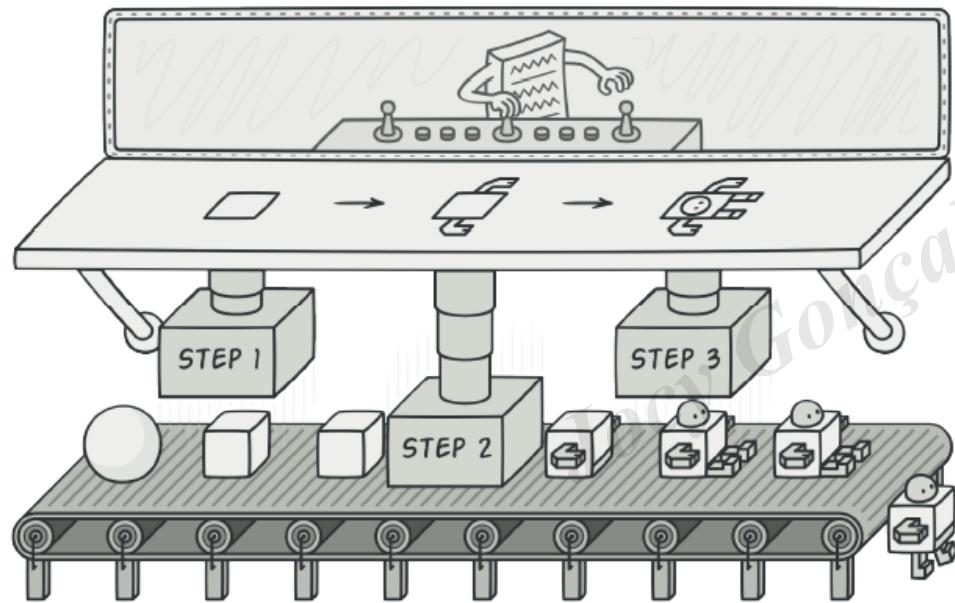
Padrão de Projeto Fábrica abstrata

Exemplo do Padrão Fábrica abstrata

```
//Determina o estilo instanciando meuEstilo  
EstiloModernoC meuEstilo = new EstiloModernoC;  
// Cria os armários de parede: Tipo determinado pela classe de  
meuEstilo  
ArmarioDeParede ArmarioDeParedel = meuEstilo.obterArmarioDeParede();  
ArmarioDeParede ArmarioDeParede2 = meuEstilo.obterArmarioDeParede();  
.  
.  
// Cria os armários de chão: Tipo determinado pela classe de  
meuEstilo  
// Cria o objeto cozinha (no estilo requerido)  
ArmarioDeChao armarioDeChao1 = meuEstilo.obterArmarioDeChao();  
ArmarioDeChao armarioDeChao2 = meuEstilo.obterArmarioDeChao();  
.  
.  
Cozinha cozinha = new Cozinha();  
Cozinha.adicionar( armarioDeParedel, . . . );  
Cozinha.adicionar( armarioDeParede2, . . . );  
.  
.  
Cozinha.adicionar( armarioDeChao1 . . . );  
Cozinha.adicionar( armarioDeChao2 . . . );  
.
```

Padrão de Projeto Construtor (do inglês Builder)

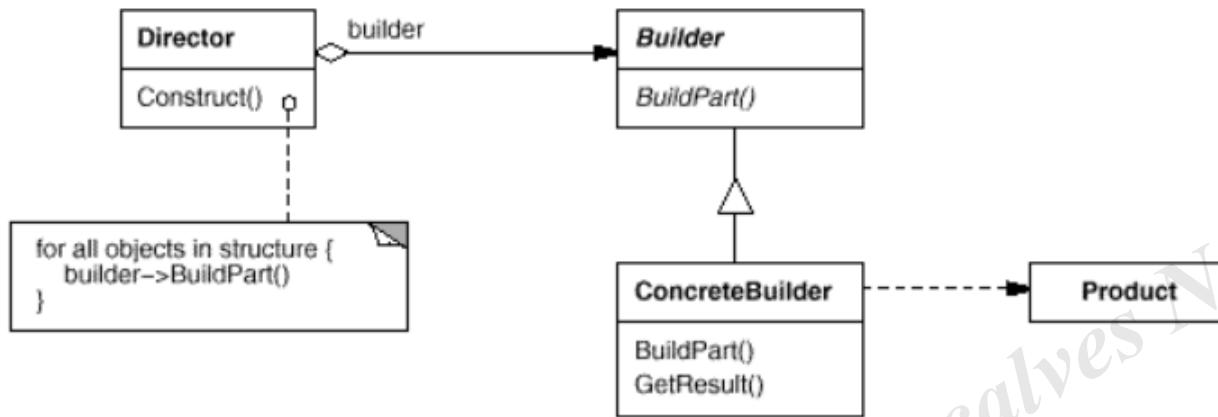
- **Classificação:** Padrão de Criação/Escopo de objetos
- **Intenção:** Separar a construção de um objeto complexo de sua representação para que o mesmo processo de construção possa criar representações diferentes.



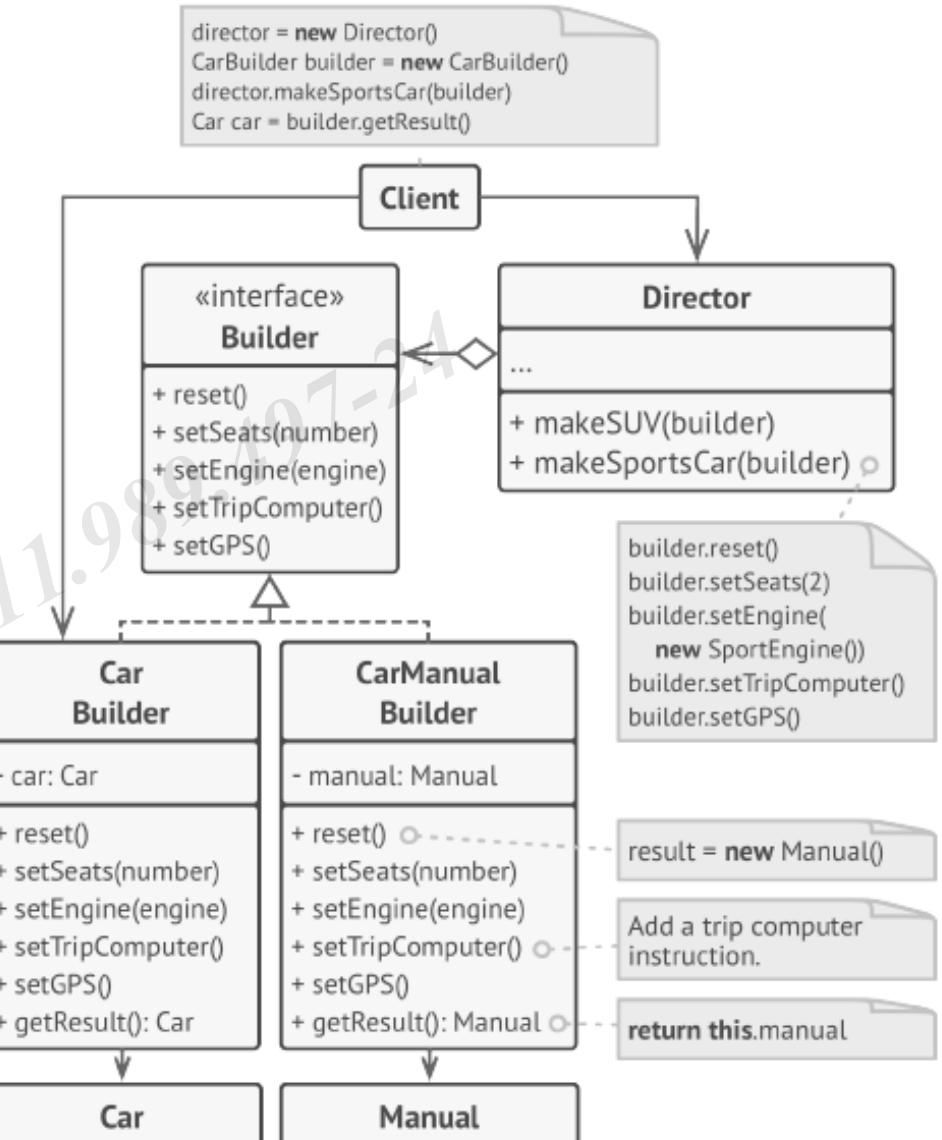
(Shvets, Alexander, 2018)

Padrão de Projeto Construtor

- Estrutura:



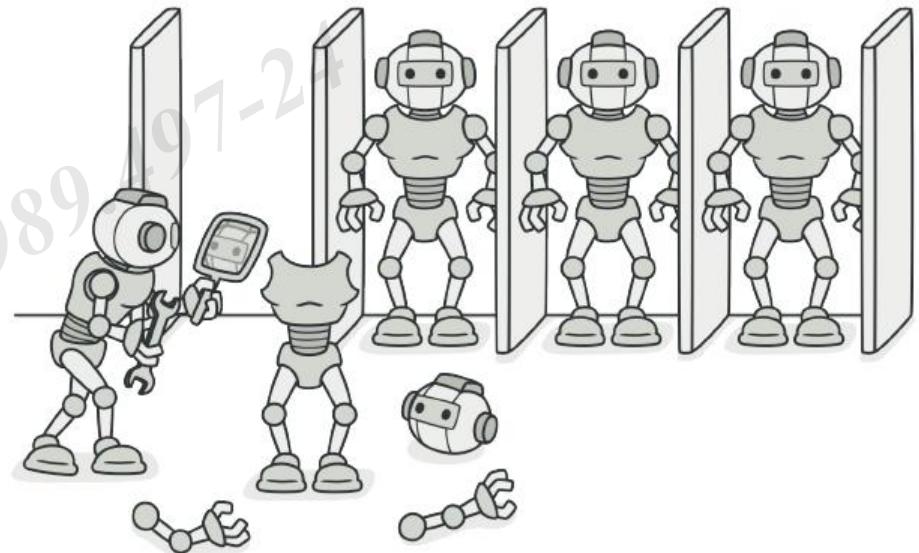
** Imagem capturada no livro versão html
publicado em CD (Gamma et al 1995)



(Shvets, Alexander, 2018)

Padrão de Projeto Protótipo (do inglês Prototype)

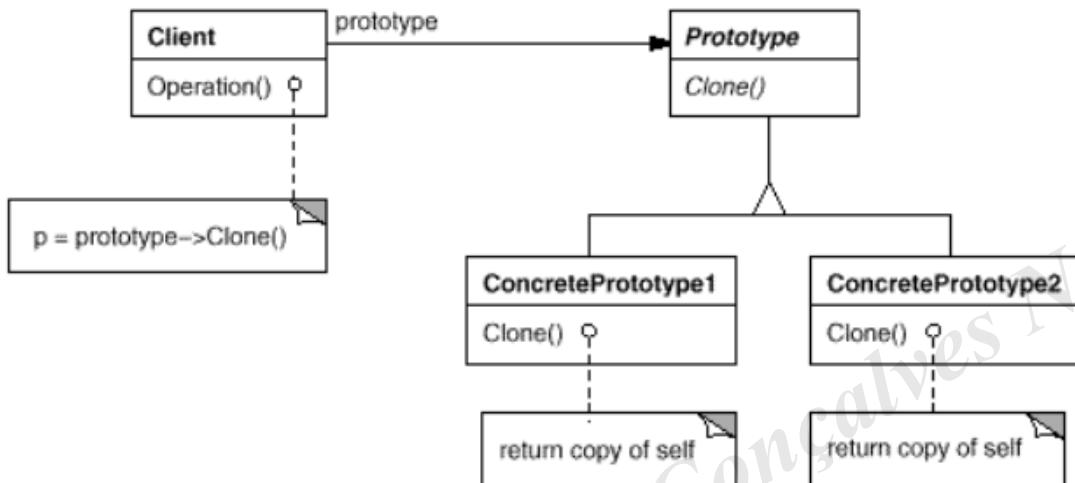
- **Classificação:** Padrão de Criação/Escopo de objetos
- **Intenção:** Especificar os tipos de objetos a serem criados usando uma instância prototípica e criar novos objetos copiando este protótipo, fazendo que o código não fique dependente da classe desses objetos
- **Aplicabilidade:** casos em que objetos são passados por meio de código de terceiros por meio de alguma interface. As classes concretas desses objetos são desconhecidas, e você não poderia depender deles mesmo se quisesse.



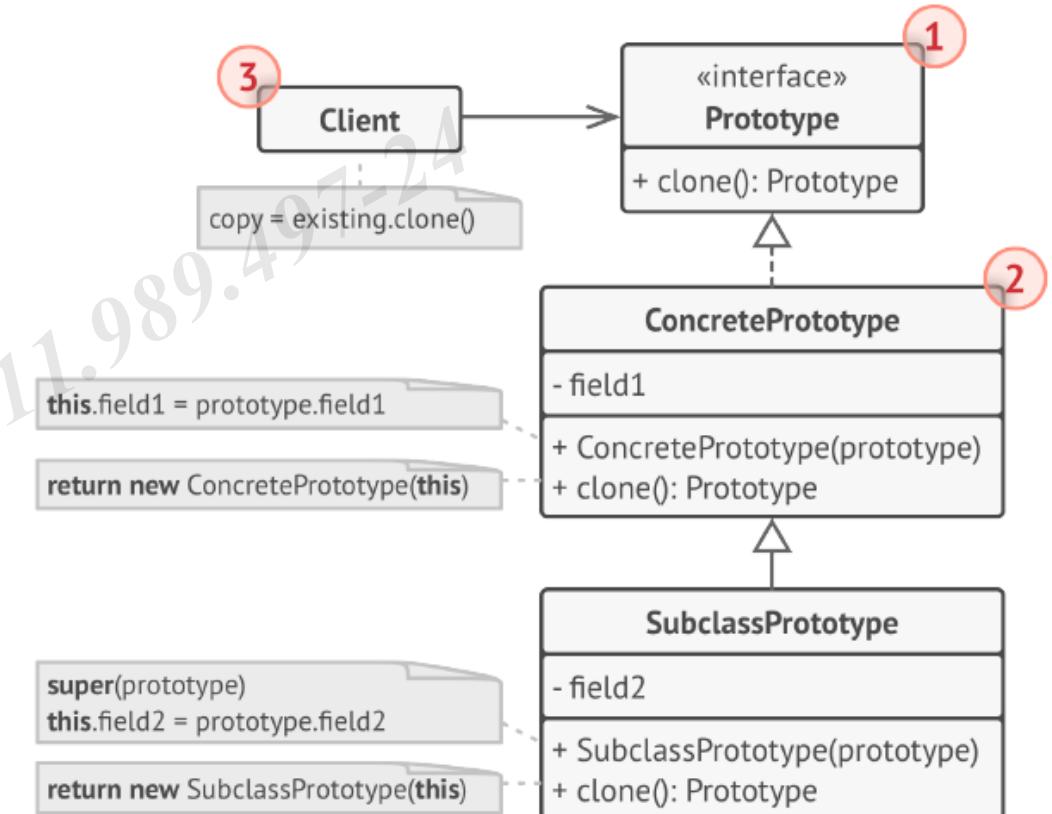
(Shvets, Alexander, 2018)

Padrão de Projeto Protótipo

- Estrutura:



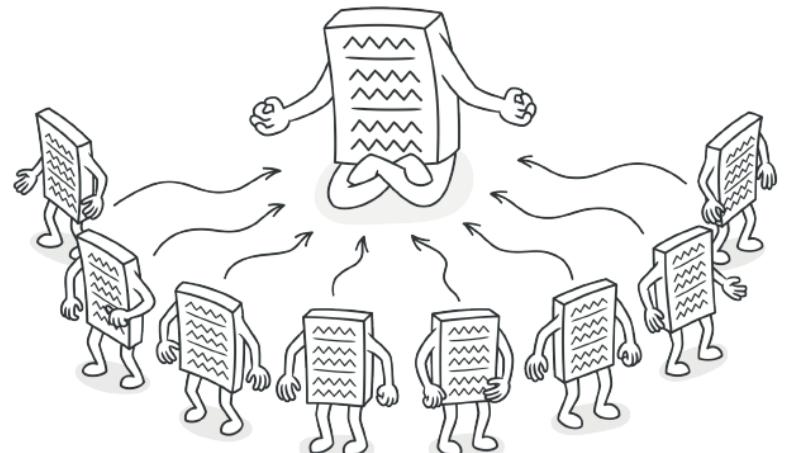
** Imagem capturada no livro versão html
publicado em CD (Gamma et al 1995)



(Shvets, Alexander, 2018)

Padrão de Projeto Objeto Unitário (do inglês Singleton)

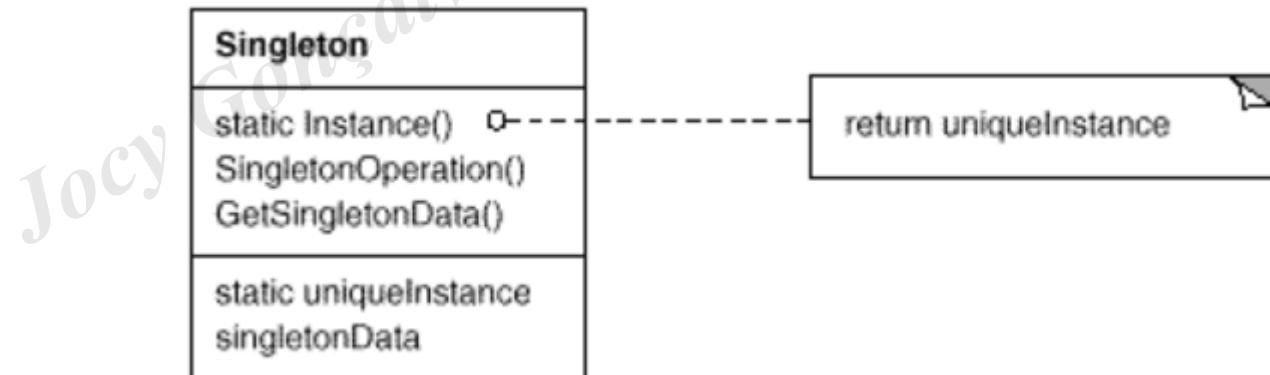
- **Classificação:** Padrão de Criação/Escopo de objetos
- **Intenção:** Garantir que uma classe tenha apenas uma instância e forneça um ponto global de acesso a ela.
- **Motivação:** Em algumas aplicações é importante garantir que algumas classes tenham exatamente uma instância, por exemplo, apenas um sistema de arquivos e um gerenciador de janelas.
 - Uma variável global poderia resolver o problema?



(Shvets, Alexander, 2018)

Padrão de Projeto Singleton

- Aplicabilidade
- Use o padrão Singleton quando:
 - deve haver exatamente uma instância de uma classe, e ela deve ser acessível aos clientes de maneira fácil.
 - quando a única instância deve ser extensível por meio de herança, e os clientes devem ser capazes de usar uma instância estendida sem modificar seu código.
- Estrutura:



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

Padrão de Projeto Singleton

- Participante:



(Braude, Eric, 2005)

- Define uma operação Instancia() (na classe) que permite aos clientes acessar a instância única.

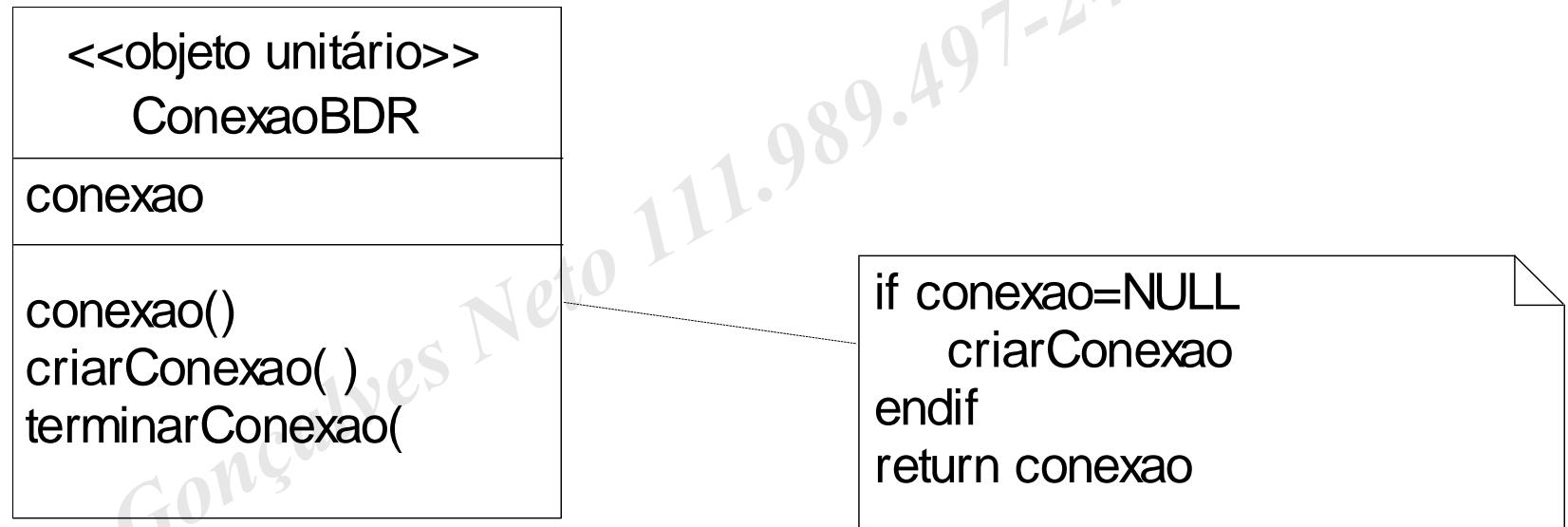
Padrão de Projeto Singleton

Exemplo do Padrão Singleton

- Ao iniciar uma aplicação, é necessário estabelecer uma conexão com a base de dados, para ter um canal de comunicação aberto durante a execução da aplicação.
- Em geral, isso é implementado por meio de uma classe Conexão (várias interfaces de BDRs existentes fornecem uma classe Connection especialmente para este fim).
- Esta conexão deve posteriormente ser fechada, no momento do término da aplicação.
- A conexão com o BDR deve preferencialmente ser única, pois se cada vez que um objeto for criado uma nova conexão for criada, inúmeros objetos existirão na memória, desnecessariamente.
- Assim, pode-se aplicar o padrão Singleton para garantir que, para cada aplicação sendo executada, uma única conexão válida com o BDR existe.

Padrão de Projeto Singleton

Exemplo do Padrão Singleton



```

// final -> evita que seja feita uma herança
public final class ConnectionPool {

    private static final String DATA_SOURCE_MYSQL = "java:comp/env/jdbc/passeLivre";
    private DataSource dataSource;           // pool de conexão com a base de dados
    private static ConnectionPool mySelf;   // referência para uma única instância dessa classe

    // construtor privado
    private ConnectionPool( DataSource dataSource ) {

        this.dataSource = dataSource;
    }
    // synchronized para evitar que mais de uma instância seja criada num sistema multithread
    public static synchronized ConnectionPool getInstance() {

        try {
            // verifica se ainda não foi criada uma única instância
            if( mySelf == null ) {

                // pega o contexto da aplicação
                Context contexto = new InitialContext();
                // pega o pool de conexões com a base
                DataSource dataSource = ( DataSource )contexto.lookup( DATA_SOURCE_MYSQL );
                // cria a única instância dessa classe
                mySelf = new ConnectionPool( dataSource );
            }
        } catch( NamingException e ) {
            System.err.println( e.getMessage() );
        }
        return mySelf;
    }

    public Connection getConnection() throws SQLException {
        return dataSource.getConnection();
    }
}

```

Exemplo do Padrão Singleton



Enquete 6 : Uso do Singleton



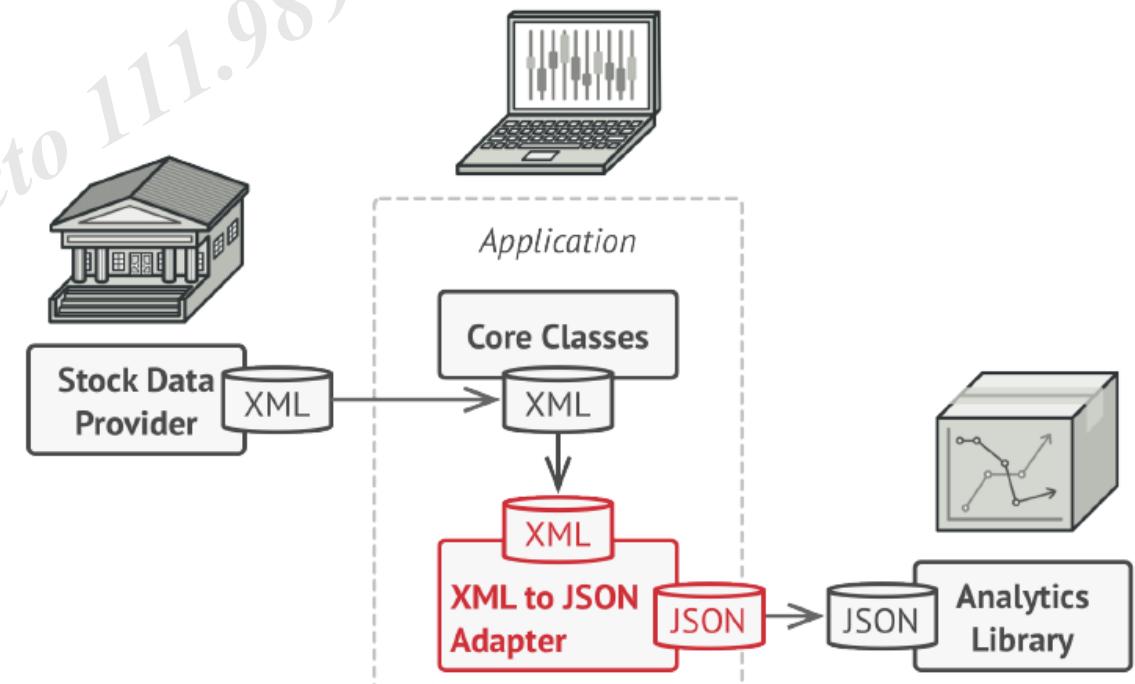
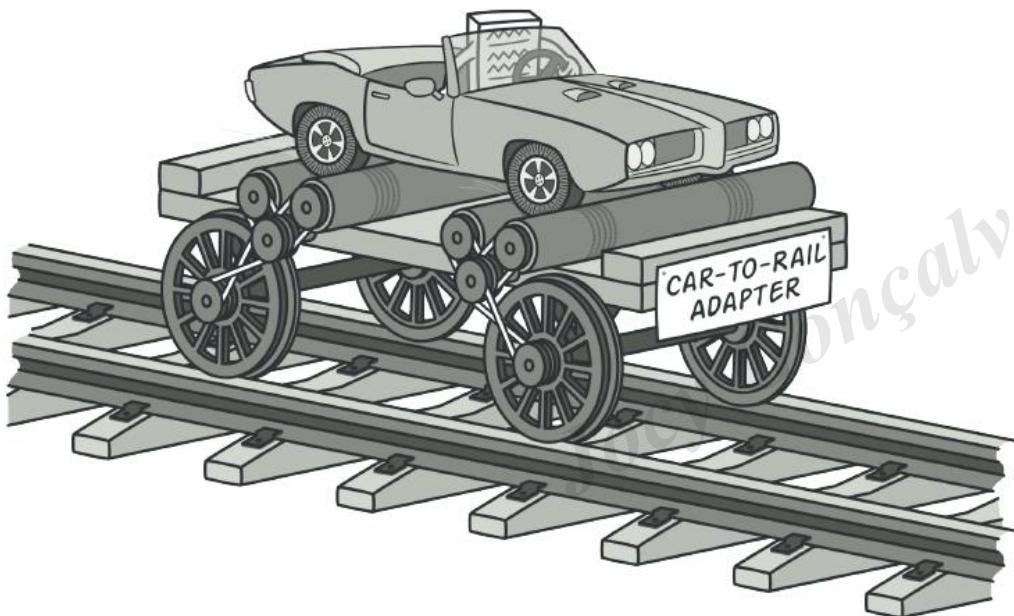
Escopo	Propósito		
	De criação	Estrutural	Comportamental
	Classe	Método-fábrica	Adaptador
Objeto	<ul style="list-style-type: none"> • Fábrica Abstrata • Construtor • Protótipo • Objeto Unitário (Singleton) 	<ul style="list-style-type: none"> • Adaptador • Ponte • Composto • Decorador • Fachada • Peso-pena • Procurador 	<ul style="list-style-type: none"> • 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 Adaptador (do inglês Adapter)

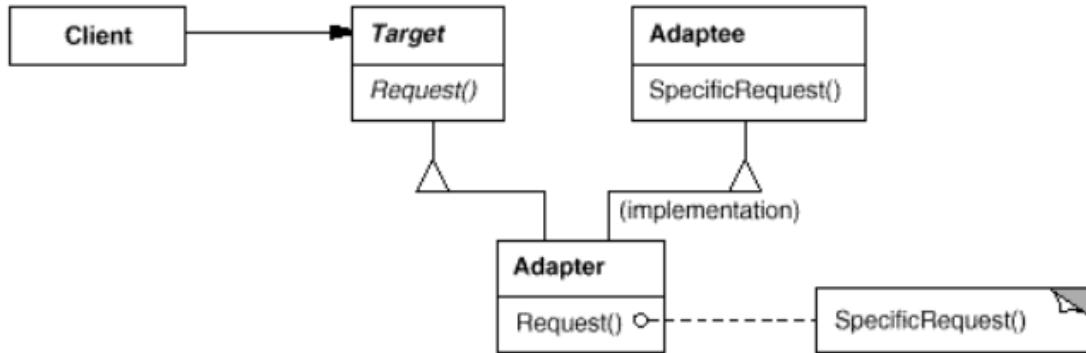
- **Classificação:** Padrão estrutural/Escopo de classes/objetos
- **Intenção:** Converter a interface de uma classe em outra interface que os clientes esperam.
O Adaptador permite que classes com interfaces incompatíveis possam trabalhar juntas



(Shvets, Alexander, 2018)

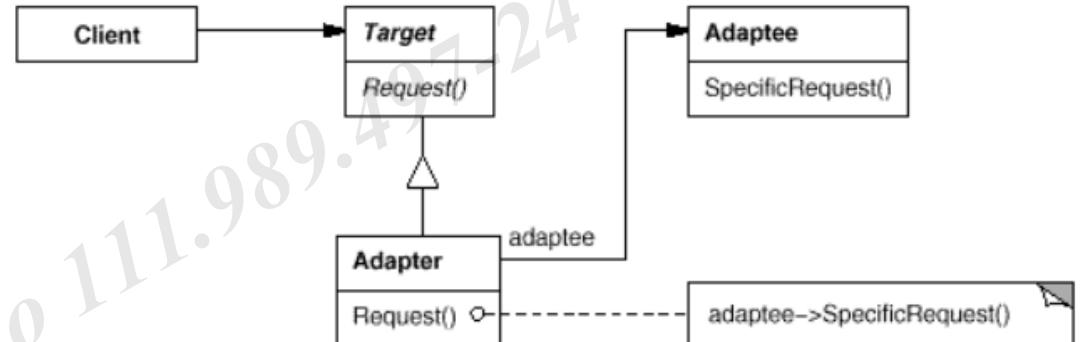
Padrão de Projeto Adaptador

- Estrutura:



Adaptador no escopo de classe

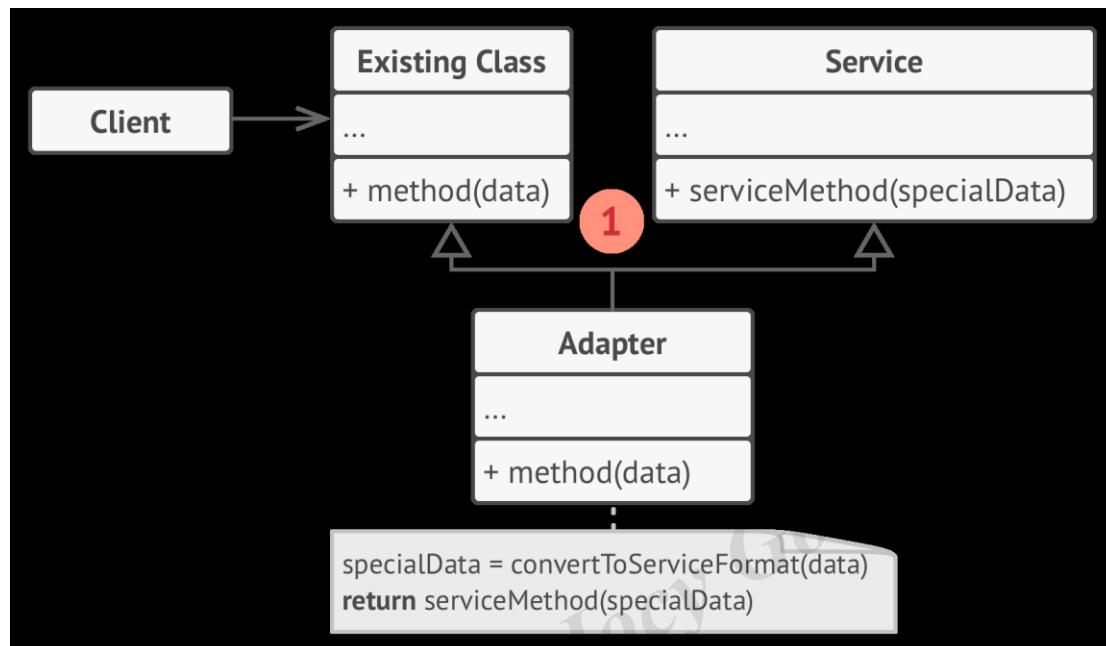
** Imagens capturadas no livro versão html
publicado em CD (Gamma et al 1995)



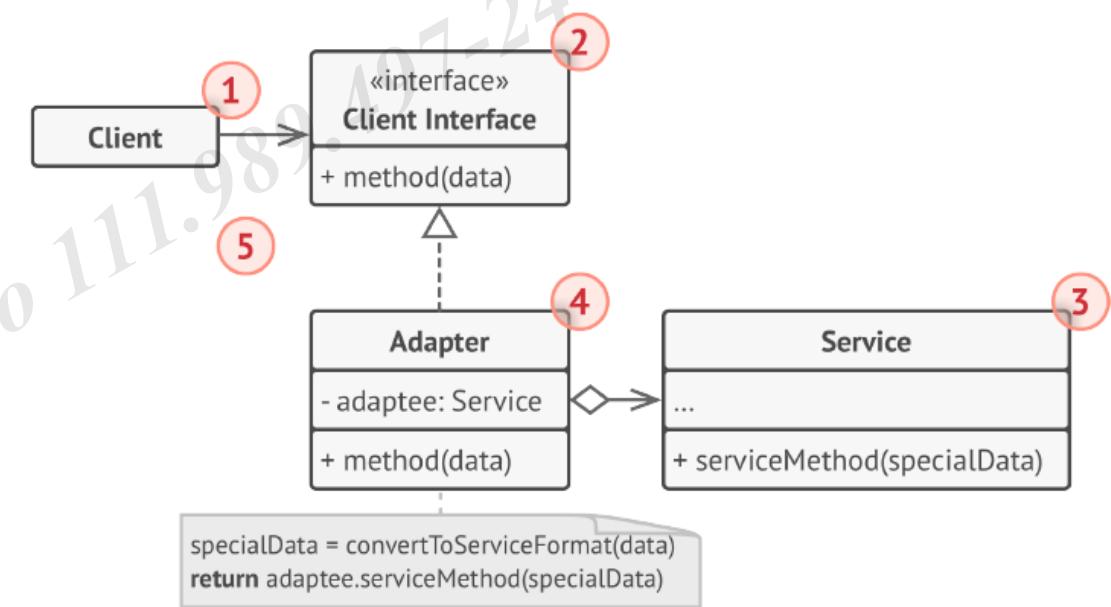
Adaptador no escopo de objeto

Padrão de Projeto Adaptador

- Estrutura:



Adaptador no escopo de classe

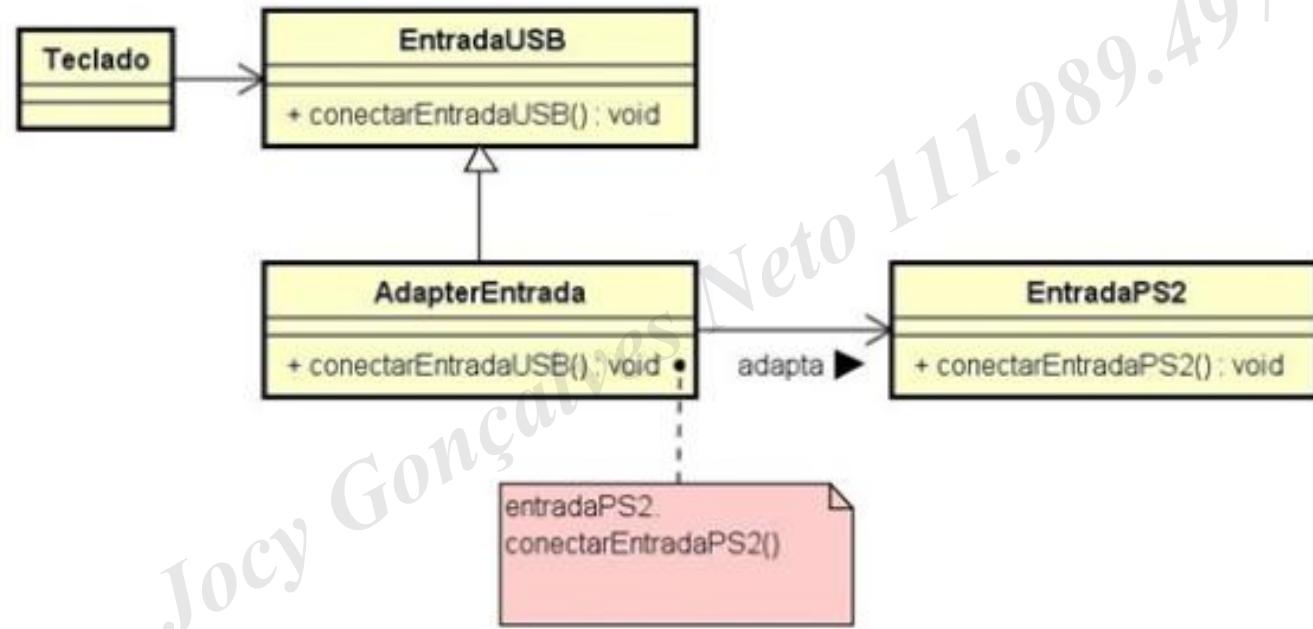


Adaptador no escopo de objeto

(Shvets, Alexander, 2018)

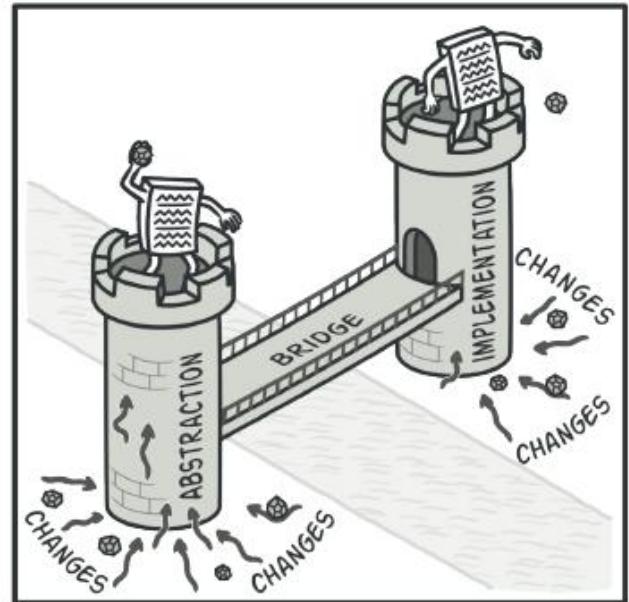
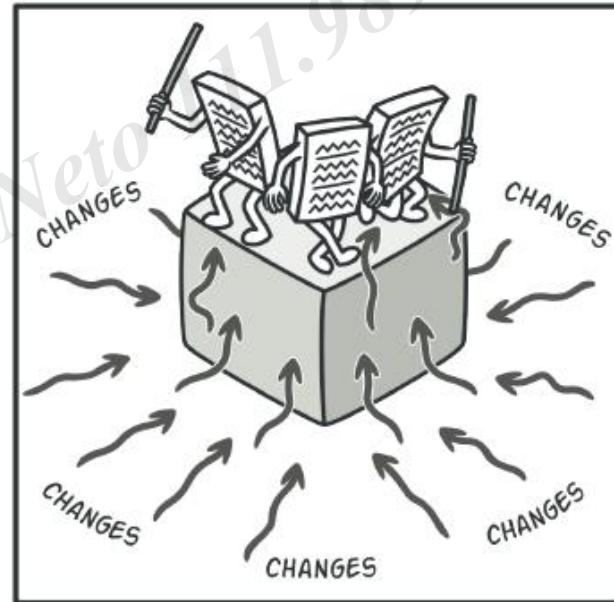
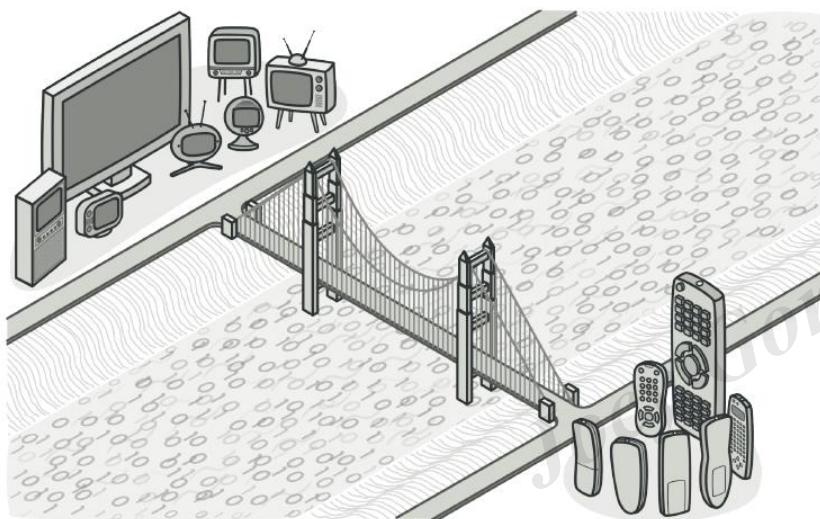
Padrão de Projeto Adaptador (do inglês Adapter)

- Exemplo:



Padrão de Projeto Ponte (do inglês Bridge)

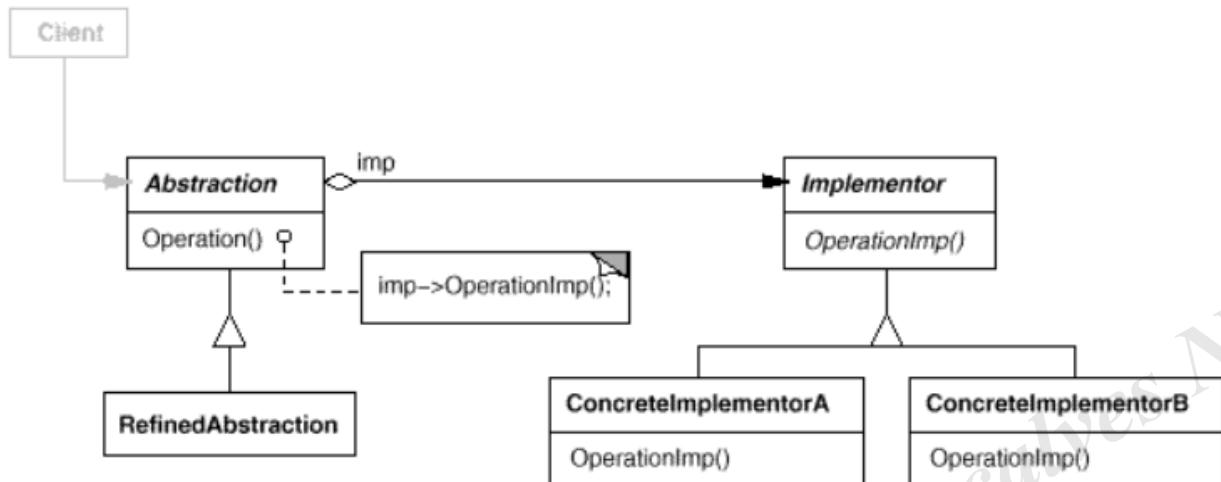
- **Classificação:** Padrão estrutural/Escopo de objetos
- **Intenção:** Desacoplar uma abstração de sua implementação para que as duas possam variar independentemente.



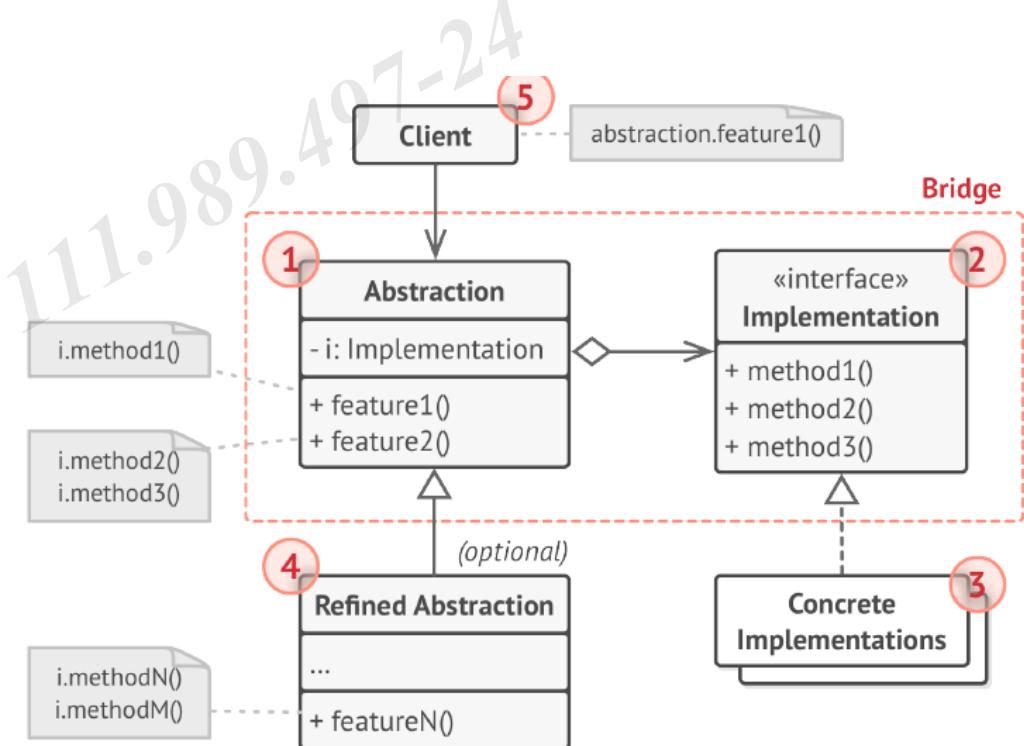
(Shvets, Alexander, 2018)

Padrão de Projeto Ponte

- Estrutura:



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



(Shvets, Alexander, 2018)

Próxima aula

- Padrões estruturais e comportamentais

		Propósito		
		De criação	Estrutural	Comportamental
Classe	Objeto	Método-fábrica	Adaptador	Interpretador Método gabarito
	<ul style="list-style-type: none">• Fábrica Abstrata• Construtor• Protótipo• Objeto Unitário (Singleton)	<ul style="list-style-type: none">• Adaptador• Ponte• Composto• Decorador• Fachada• Peso-pena• Procurador	<ul style="list-style-type: none">• Cadeia de Responsabilidade• Comando• Iterador• Mediador• Memento• Observador• Estado• Estratégia• Visitador	

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.



ANEXOS
(caso sobre tempo)

Conferências sobre Padrões - PLoP

- Submissão de artigos: processo de “shepherding”
- Conferência: Sessões de “workshop do autor – *writers' workshop*”, intercaladas por “jogos”
- Grupos com 6 a 8 autores e 6 a 8 não autores, 1 líder e 1 secretário.



• Porto das Dunas - Ceará - 2004



• Campos do Jordão – SP - 2005

Conferências sobre Padrões - PLoP

- Writers' workshop:
 - Líder pede a alguém para fazer um resumo do padrão
 - Autor tem 5 minutos para ler trechos do artigo para os demais participantes, que já leram de antemão.
 - Autor “congela”
 - 15 minutos: demais participantes levantam pontos fortes do padrão
 - 15 minutos: demais participantes apresentam sugestões para melhoria
 - Autor “descongela” e tem mais 10 minutos para perguntas e respostas.
 - Líder encerra sessão



Conferências sobre Padrões - PLoP

- Conferências sobre padrões
 - Primeira conferência PLoP – 1994 - EUA
 - Outras conferências:
 - EuroPLoP – Europa
 - ChiliPLoP – EUA (Arizona)
 - KoalaPLoP – Austrália
 - MensorePLoP – Japão
 - VikingPLoP - Escandinávia
 - No Brasil, depois estendido para América Latina:
SugarloafPLoP!!!



PLoP EUA

- Allerton House – Monticello – Illinois - EUA



1998/1999/2001



EuroPLoP

- EuroPLoP - Alemanha



SugarloafPLoP

- 2001: 1st Latin-American Conference on Pattern Languages of Programs (Rio de Janeiro)
- 2005: 5th Latin-American Conference on Pattern Languages of Programs (Campos do Jordão – SP)
- 2012: 9th Latin-American Conference on Pattern Languages of Programs (Natal – RN)



SugarloafPLoP

- 2016: 11th Latin-American Conference on Pattern Languages of Programs (Buenos Aires - Argentina)
- 2018: 12nd Latin-American Conference on Pattern Languages of Programs (Valparaiso – Chile)
- 2022: Juntou-se como trilha do PLoP (online)
- Próxima?



Obrigada!

[Profa. Dra. Rosana T. Vaccare Braga](#)

www.linkedin.com/in/rosana-braga-13778912