Padrões de desenho aplicados em programação por objectos

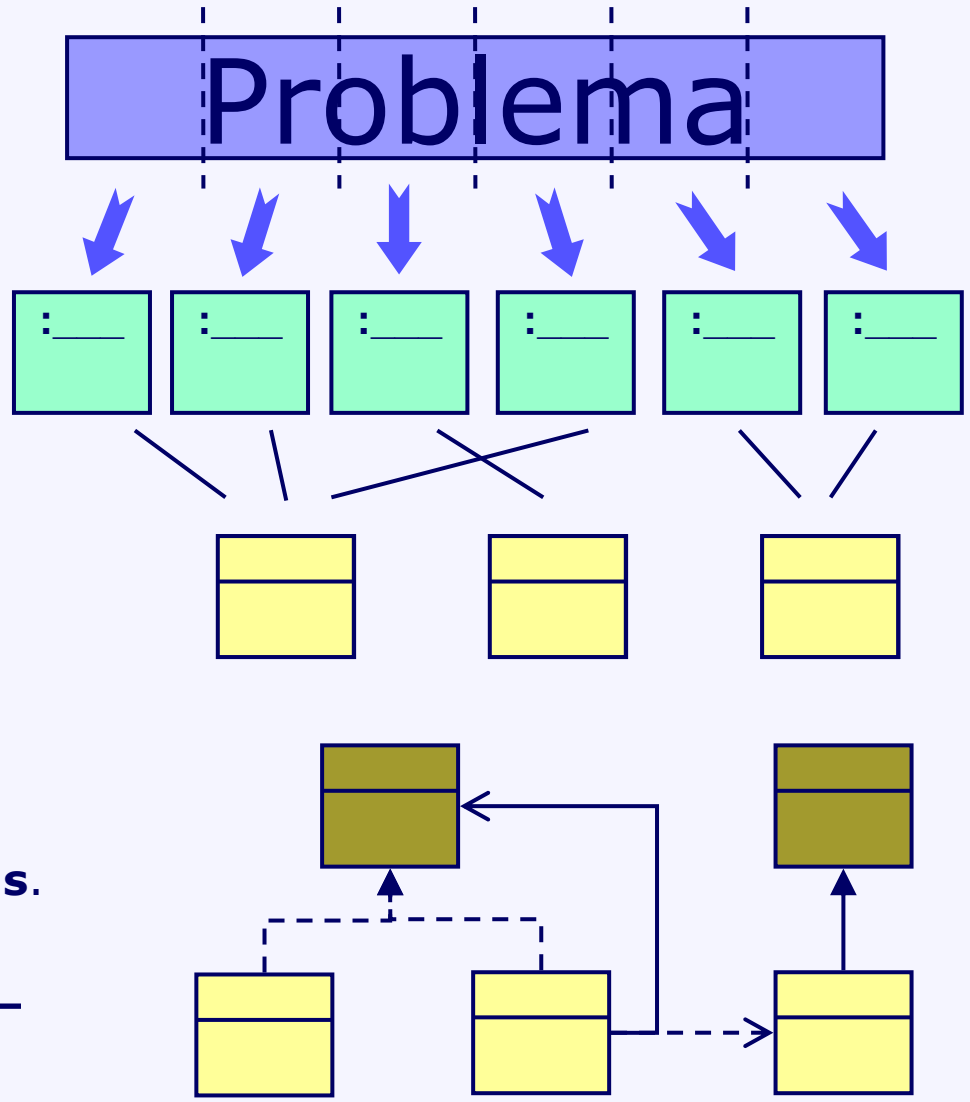
Fernando Miguel Carvalho
Centro de Cálculo
Instituto Superior de Engenharia de Lisboa, DEETC
mcarvalho@cc.isel.ipl.pt

O que fazem os programadores experientes?

Desenho em *Object Oriented*

O que fazem os programadores **experientes**?

- Não resolver todos os problemas de raiz;
- Encontrar os objectos pertinentes;
- Decompor os objectos em tipos, na granularidade **correcta**;
- Desenhar hierarquias e relações **correctas**.



Experiência

Pretende-se que os padrões de desenho ajudem a:

➔ tornar a **experiência** de um programador útil a uma comunidade de programadores.

Como?

Acções:		**Resultados:**
Registar	⇨	Reutilizar
Divulgar	⇨	Acessível
Evitar soluções	⇨	Não comprometer
Melhor documentação	⇨	Manutenção e evolução

Padrão de Desenho

Conceito:
➔ Um padrão é uma **solução** para um **problema** num dado **contexto**.

Contexto:
- Situação a que se aplica o padrão.
- Deve ser uma **situação recorrente**.

Exemplo:
colecções de objectos.

Problema:
- **Objectivo** a atingir nesse contexto.
- Conjunto de **constrangimentos** que ocorrem nesse contexto.

Exemplo:
iterar sobre os elementos de uma colecção sem expor a sua implementação.

Solução:
- **Desenho genérico** que qualquer um possa aplicar para resolver um **problema** com um conjunto de **constrangimentos** e um **objectivo** comum.

Exemplo:
encapsular o iterador numa classe separada.

Padrão de Desenho... Requisitos

Nem todas as solução podem-se tornar num padrão!

1. Uma solução para se tornar num padrão necessita de se aplicar a **problemas recorrentes**.

2. Tem de ter uma descrição suficientemente **genérica** e **precisa**, de maneira a que possa ser **adaptado a outros problemas com as mesmas características**.

- Os padrões não são leis, nem regras estáticas.
- **Os padrões são linhas orientadoras que podem ser adaptadas para servir determinadas necessidades**.

Padrão de Desenho...

contexto:
- passagens longas e estreitas;

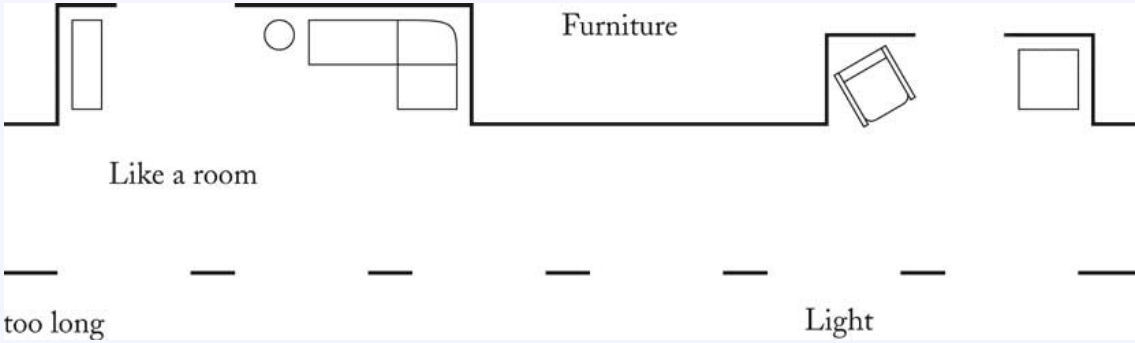
problema:
- corredores compridos são deprimentes e desconfortáveis;

constrangimentos:
- comprimento;
- falta de luz.

objectivo: evitar a ansiedade.

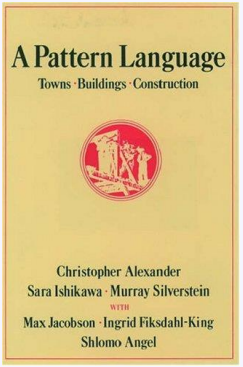
Desenho:
- evitar áreas estreitas;
- intercalar espaço amplos;
- aparência de uma sala;
- aplicar janelas.

Solução



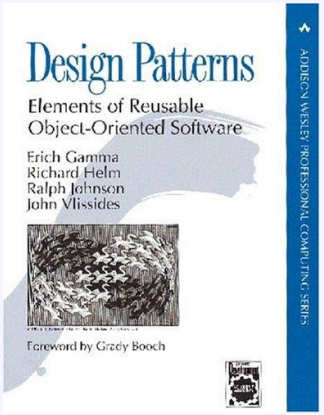
Padrões de Desenho… Definição

- *Christopher Alexander*, professor de arquitectura em Berkeley, foi o primeiro autor a descrever padrões para a arquitectura de áreas habitáveis como casas, edifícios e cidades no livro: "***A Pattern Language: Towns Buildings Constructions***" (Alexander et al. 1977).



"Cada padrão descreve um **problema que ocorre sistematicamente** num **determinado contexto** e descreve os **aspectos chave** da solução, de forma a que esta possa ser reutilizada um número indeterminado de vezes sem que em nenhuma delas seja implementado da mesma forma."

- O livro "***Design Patterns (Gamma et al. 1995)***" foi o primeiro a catalogar de forma sistemática e consistente 23 dos padrões de desenho mais utilizados (com sucesso) em programação OO.



- Este livro tornou-se a principal referência dos **padrões de desenho fundamentais** em programação OO e também é identificado de modo informal na comunidade científica por **GoF**, ou ***Gang-of-Four*** (por terem sido quatro os autores do livro).

Padrões de Desenho… descrição do GoF

- Nome
- Classificação
- Objectivo
- Outros nomes do padrão
- Motivação
- Aplicabilidade
- Estrutura
- Participantes
- Colaboração
- Consequências
- Implementação
- Código exemplo
- Utilizações conhecidas
- Padrões relacionados

Padrão *Template Method*

3 slides com:

- Estrutura: diagrama de classes UML exemplificativo do padrão.
-

Nome do Participante	Descrição
Nome Genérico `Ex: NomeDaClasse`	•
...	

-

Característica	Descrição
Nome	
Categoria	<Finalidade> – <Âmbito de Acção>
Objectivo	
Aplicabilidade	• •
Nome alternativo	

Padrões de Desenho… Classificação

O esquema de classificação usado no GoF está organizado em 3 categorias relacionadas com a **finalidade** dos padrões:

- **Padrões de Criação**: envolvem a instanciação de objectos e oferecem uma forma de desacoplar o cliente dos objectos que o cliente necessita de instanciar.
- **Padrões Estruturais**: compõem classes e objectos para definir novas estruturas ou novas funcionalidades.
- **Padrões Comportamentais**: concentram-se na forma como os tipos e objectos interactuam e delegam as responsabilidades entre si.

A classificação dos padrões oferece:

➔ organização e agilidade na pesquisa;

➔ termo de comparação de um padrão dentro da sua categoria.

Padrões de Desenho… Classificação...

Dentro de cada uma das categorias anteriores é usual encontrar os padrões classificados por um **segundo atributo** relacionado com o **âmbito de acção**:

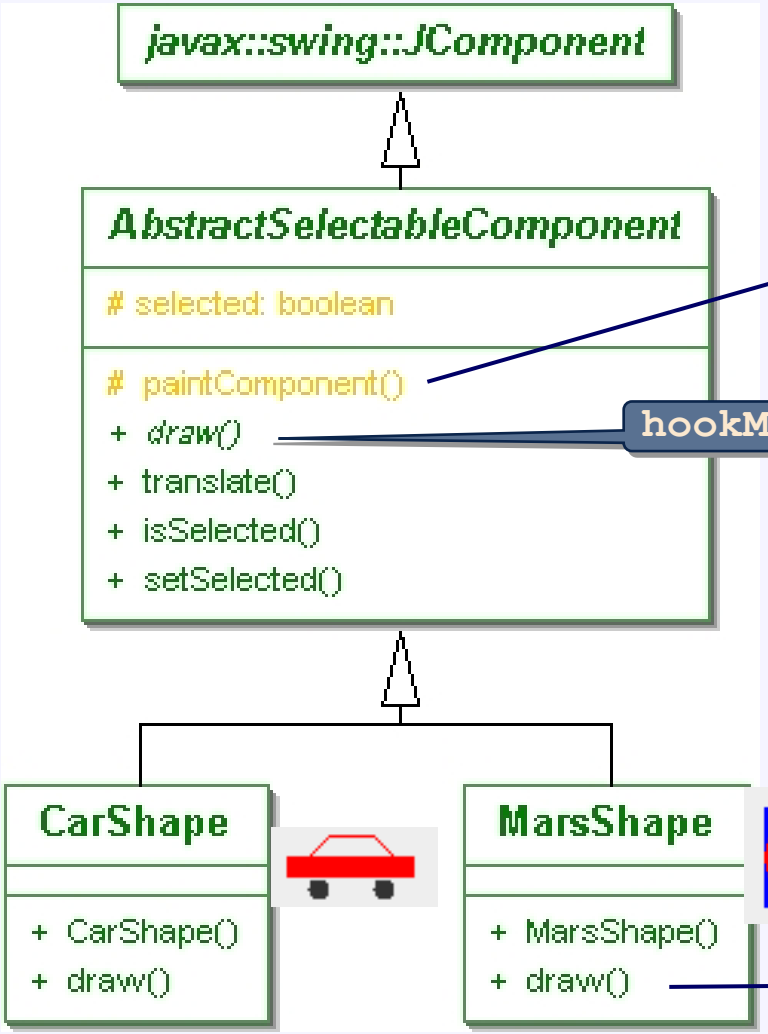
- **Classes**: as relações entre os tipos são definidas através de **herança** e estabelecem-se **estaticamente** em **tempo de compilação**.
- **Objectos**: as relações entre os objectos são definidas por **composição** e podem mudar **dinamicamente** em **tempo de execução**.

Padrões de Desenho… Classificação...

		Propósito		
		Criação	Estrutura	Comportamento
Âmbito de acção	Classes	Factory Method	Adapter (class)	Interpreter Template Method
	Objectos	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Momento Observer State Strategy Visitor

Padrão *Template Method*

Exemplo




```
Graphics2D g2 = (Graphics2D) g;
draw(g2);
if(isSelected()){
  g2.setColor(Color.GRAY);
  g2.draw3DRect(0, 0, getWidth(), getHeight(), true);
}
```

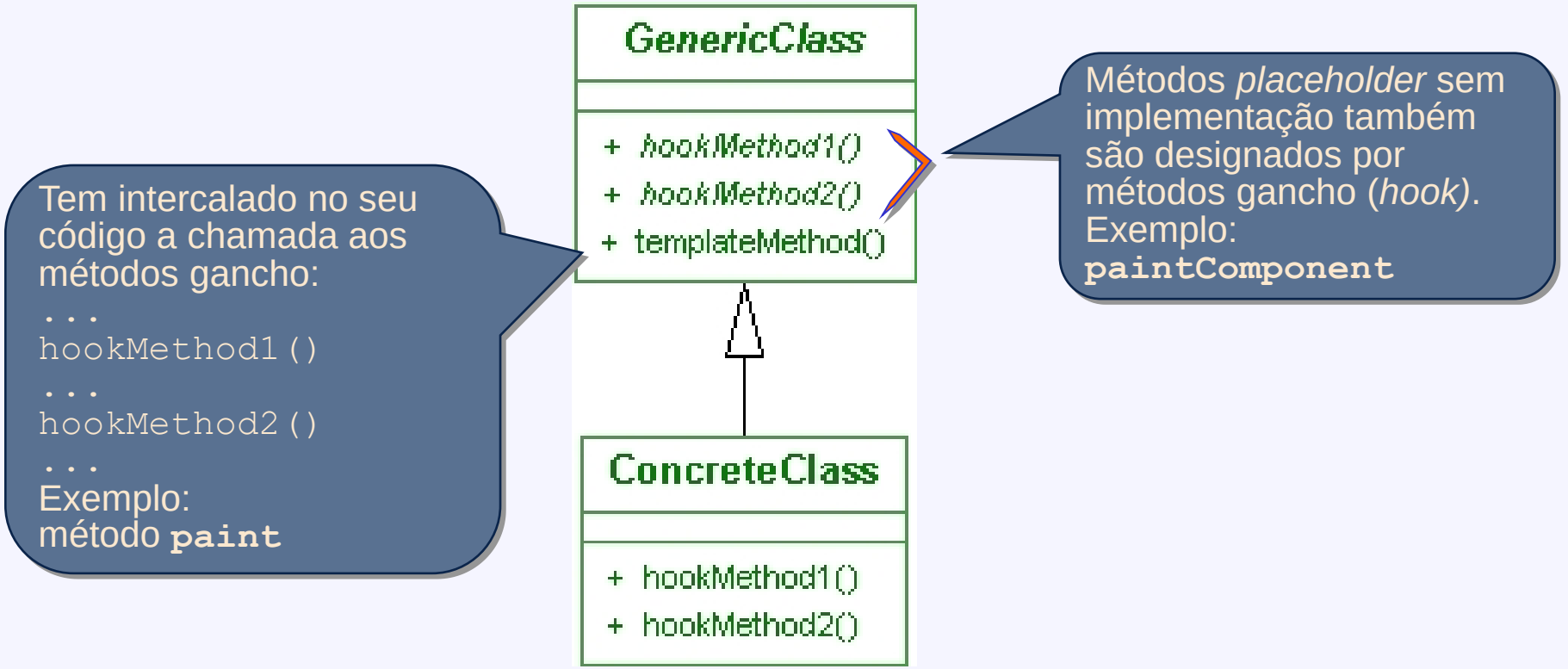
```
g.setColor(Color.BLUE);
g.fillRect(0, 0, getWidth(), getWidth());
g.setColor(Color.RED);
g.fillOval(0, 0, getWidth(), getWidth());
```

Padrão *Template Method* - Participantes

Nome do Participante	Descrição
`GenericClass (AbstractSelectableComponent)`	• Define os métodos abstractos (*hook* ou *primitive ou placeholder*) que as subclasses redefinem para implementar as partes variáveis de um algoritmo; • implementa o método template que define o esqueleto do algoritmo recorrendo aos métodos abstractos
`ConcreteClass (Carshape e Marsshape)`	Implementa os métodos abstractos com as partes **variáveis** do algoritmo definido no método template

Padrão *Template Method*

A classe **`JComponent`** segue o padrão *template method* na forma como deve ser especificado o desenho do seu conteúdo.

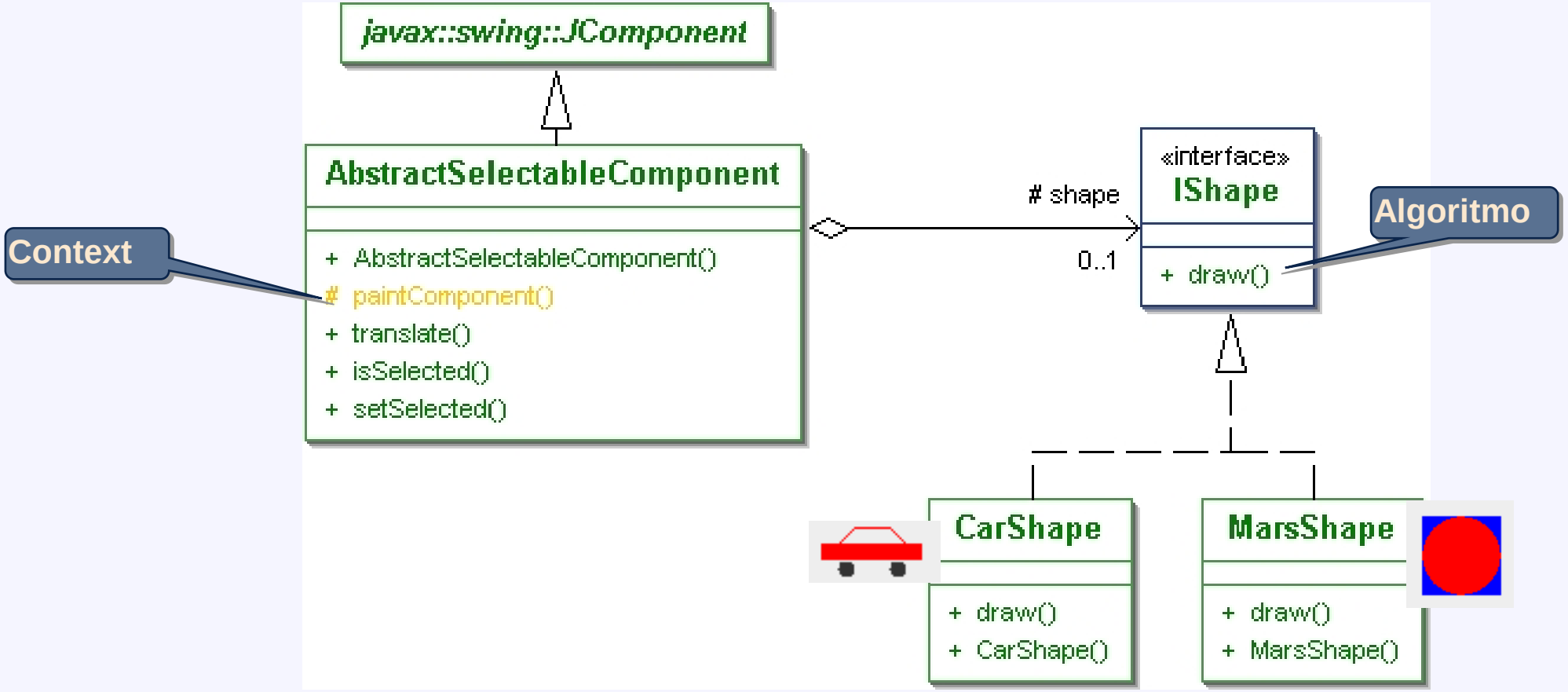


Padrão *Template Method*

Característica	Descrição
Nome	*Template Method*
Categoria	Comportamento - Classe
Objectivo	Definir o esqueleto de um algoritmo num método, permitindo que as subclasses redefinam alguns dos passos.
Aplicabilidade	• Para implementar a parte invariante do algoritmo uma só vez e permitir que as subclasses implementem o comportamento variável • Em *refactoring* para colocar o comportamento comum na superclasse e evitando o código repetido nas subclasses.
Nome alternativo	

Padrão *Strategy*

Exemplo


javax::swing::JComponent
AbstractSelectableComponent
+ AbstractSelectableComponent()
# paintComponent()
+ translate()
+ isSelected()
+ setSelected()
Context
shape
0..1
«interface»
IShape
+ draw()
Algoritmo
CarShape
+ draw()
+ CarShape()
MarsShape
+ draw()
+ MarsShape()

Padrão *Strategy* - Participantes

Nome do Participante	Descrição
`Strategy (IShape)`	Define uma interface comum a todos os algoritmos
`ConcreteStrategy (Marsshape, Carshape)`	Implementa os algoritmos através da interface da `Strategy`.
`Context (AbstractSelectableComponent)`	Mantém referência para uma ou mais instâncias de `Strategy`.

NOTAS:

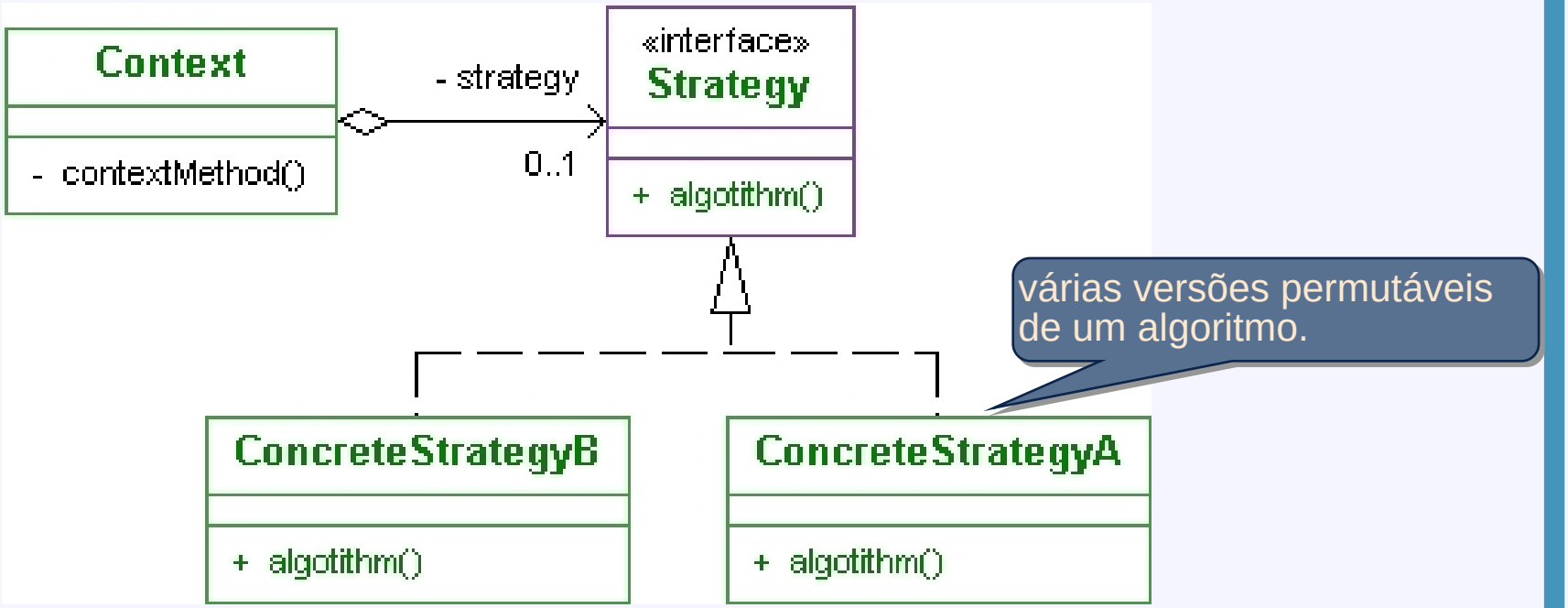
- Este padrão é uma variante do *Template Method*, onde os métodos de *hook* e o template residem em classes distintas
  - Os métodos abstractos declarados em `Strategy` são os *hooks*;
  - Os métodos em `Context` que invocam os métodos de `Strategy` são os *templates*.

Padrão *Strategy*

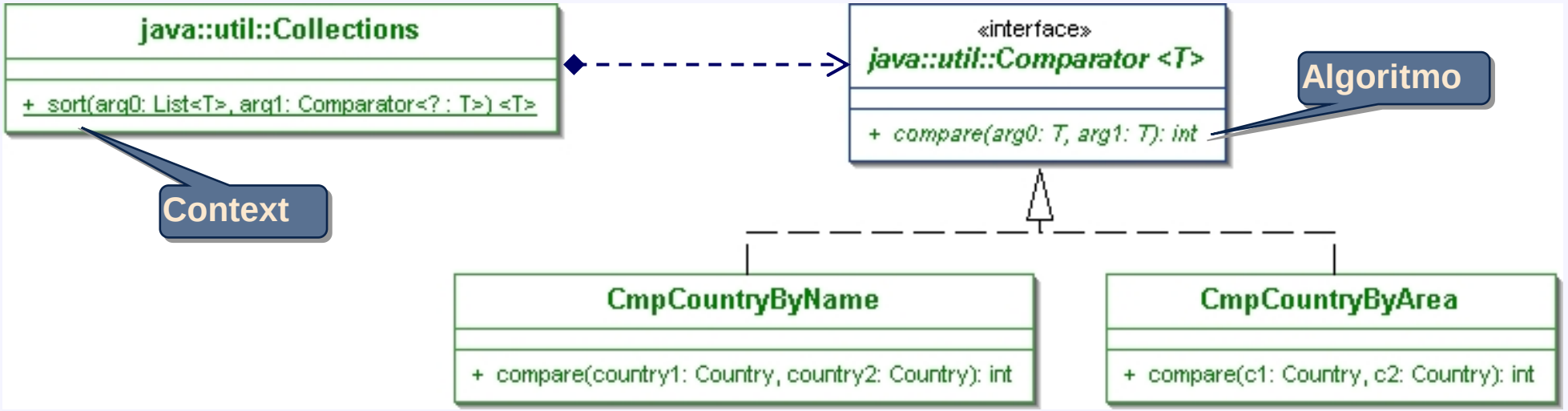
Característica	Descrição
Nome	*Strategy*
Categoria	Comportamento - Objectos
Objectivo	Definir e encapsular uma família de algoritmos, e torná-los permutáveis.
Aplicabilidade	• Classes relacionadas diferem apenas no seu comportamento; • São necessárias **várias versões de um algoritmo**; • Um algoritmo utiliza dados que devem ser desconhecidos dos clientes; • Uma classe define vários comportamentos que se manifestam em múltiplas instruções de condição nos seus métodos;
Nome alternativo	

Padrão *Strategy* - Estrutura

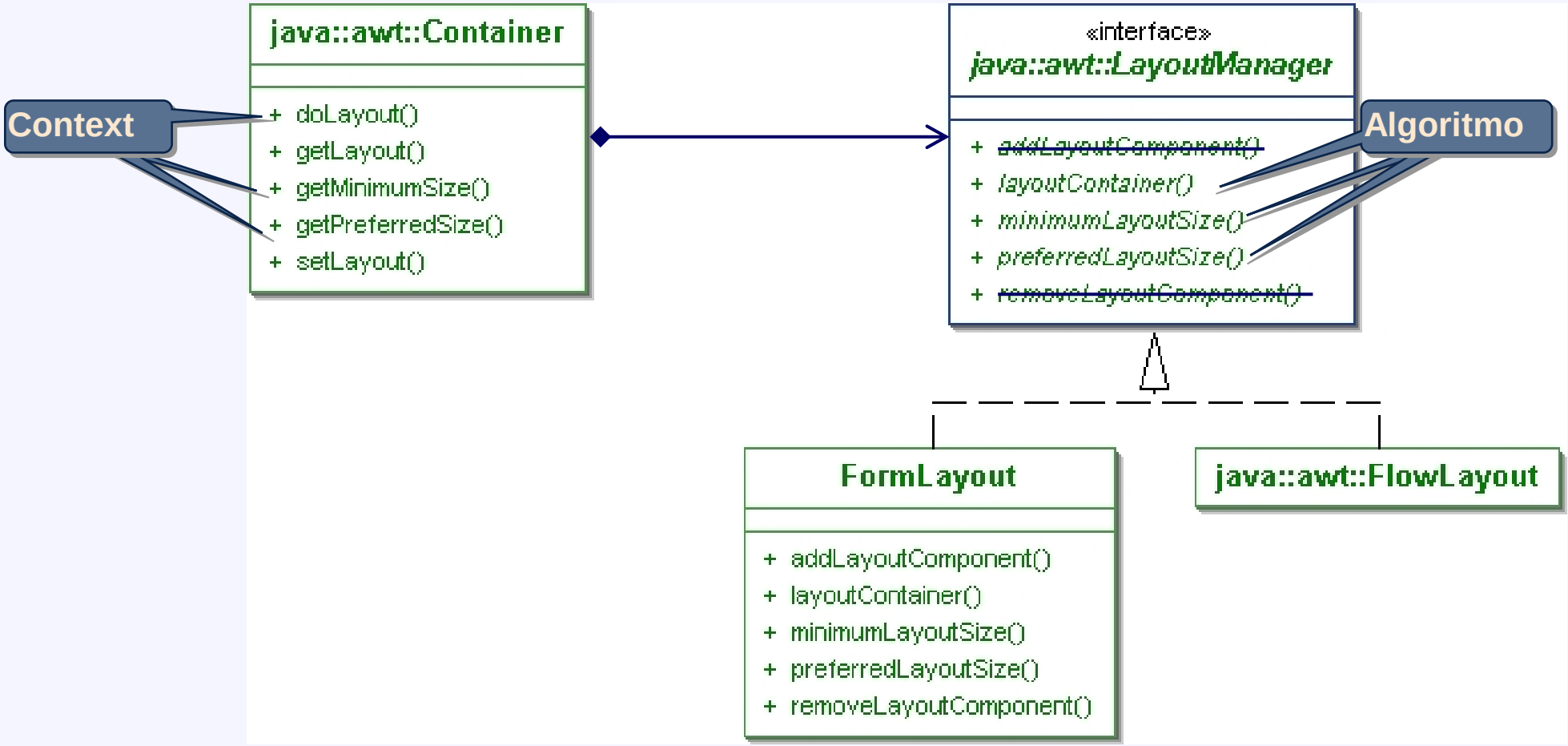
- Este padrão é uma variante do *Template Method*, onde os métodos de *hook* e o *template* residem em classes distintas



Exemplo2


java::util::Collections
+ sort(arg0: List<T>, arg1: Comparator<? : T>) <T>
Context
«interface»
java::util::Comparator <T>
+ compare(arg0: T, arg1: T): int
Algoritmo
CmpCountryByName
+ compare(country1: Country, country2: Country): int
CmpCountryByArea
+ compare(c1: Country, c2: Country): int

Exemplo 3


java::awt::Container
+ doLayout()
+ getLayout()
+ getMinimumSize()
+ getPreferredSize()
+ setLayout()
«interface»
java::awt::LayoutManager
+ addLayoutComponent()
+ layoutContainer()
+ minimumLayoutSize()
+ preferredLayoutSize()
+ removeLayoutComponent()
Context
Algoritmo
FormLayout
+ addLayoutComponent()
+ layoutContainer()
+ minimumLayoutSize()
+ preferredLayoutSize()
+ removeLayoutComponent()
java::awt::FlowLayout