



Módulo 15





BackEnd Java



Rodrigo Pires





Padrões de Projetos - Parte 1



Padrões de Projetos

O objetivo dos padrões de projetos, são tornar componentes reutilizáveis que facilitam a padronização, que permita agilidade para as soluções de problemas recorrentes no desenvolvimento do sistema.





Padrões de Projetos


Existem 2 Padrões de Projetos ou Design Patternes.

- Padrões GRASP
- Padrões GOF

Os Padrões de Projetos GOF foram criados em 1995 pelos profissionais:

- Erich Gamma
- Richard Helm
- Ralph Johnson
- John Vlissides

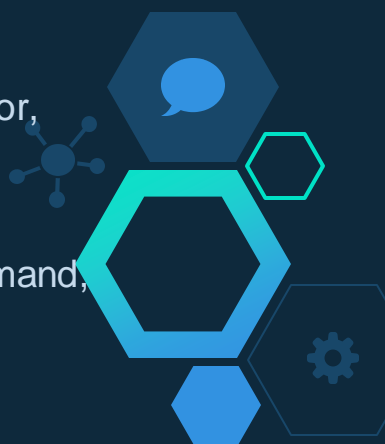
Os profissionais foram batizados com o nome “Gangue dos Quatro” (Gang of Four ou GoF)





Padrões GOF

Esses padrões tem como objetivo, solucionar problemas comuns de softwares que tenham algum envolvimento a **orientação a objetos**. São formados por três grupos exibidos abaixo:

- **Padrões de criação:** Factory Method, Abstract Factory, Singleton, Builder, Prototype.
 - **Padrões estruturais:** Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy;
 - **Padrões comportamentais:** Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template, Method, Visitor.
- 



Padrões GOF criação

Os padrões desse tipo, exigem um tratamento de como os objetos (classes) são criados, para atenderem às diversas necessidades.

- Factory Method
- Singleton
- Builder
- Abstract Factory
- Prototype.



A decorative pattern of hexagons in various shades of blue and cyan on the left side of the slide. Some hexagons contain icons: a lightbulb, a thumbs up, a smartphone, a magnifying glass, and a gear. A network diagram with a central node and six peripheral nodes is also visible.

2

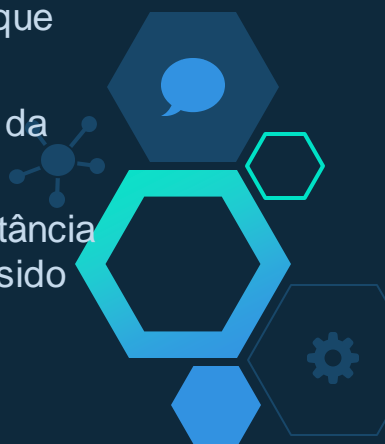
Singleton

Padrão de criação



Singleton

Usado quando desejado, que uma classe tenha apenas uma instância na aplicação. Abaixo, mostra alguns aspectos que devem ser cuidados ao criar esse padrão.

- O construtor da classe fica como privado (private), sendo que não pode ser instanciada para fora da própria classe.
 - A classe é final, pois não permite a criação de subclasses da própria classe.
 - O acesso é permitido através do método que retorna a instância única da classe, ou faz a criação de uma, caso não tenha sido criada.
- 

Singleton

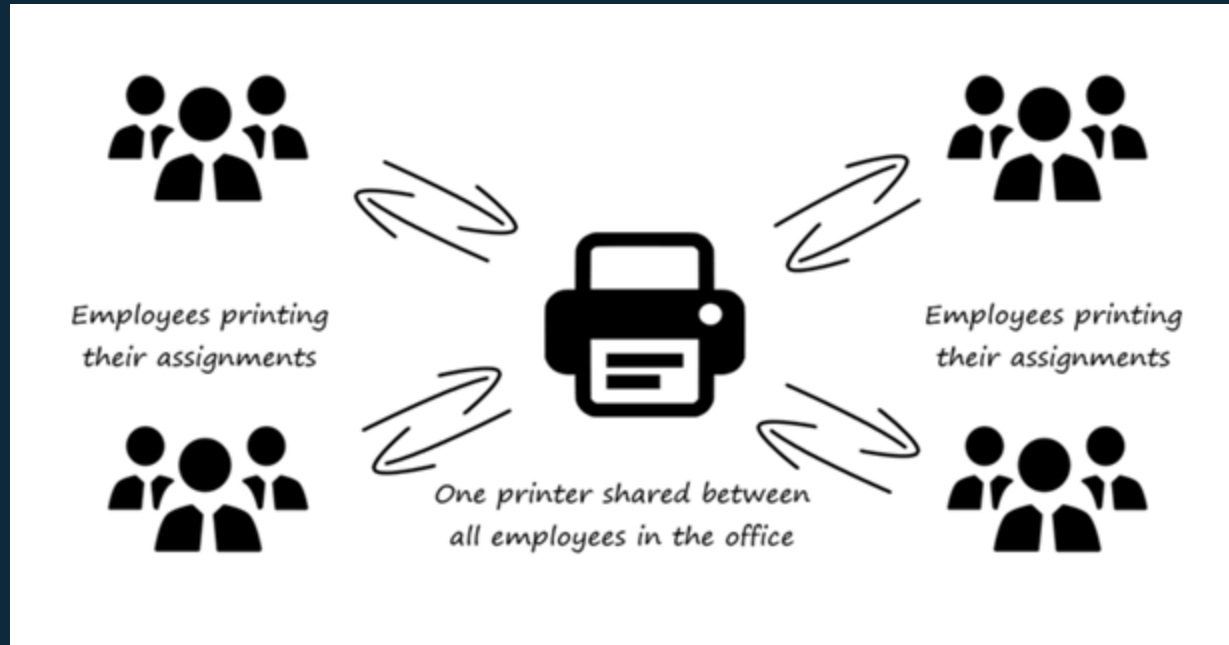


Figura 1



Singleton

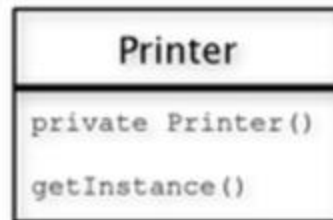


Figura 2



A decorative graphic on the left side of the slide. It features a large central hexagon with a blue-to-teal gradient, containing the number '3'. Surrounding this central hexagon are several smaller hexagons of varying shades of blue and teal. Some of these smaller hexagons contain white icons: a lightbulb (top left), a thumbs-up (top left, below the lightbulb), a smartphone (bottom left), a magnifying glass (bottom center), and a gear (bottom center). There is also a network-like icon with a central node and five connecting lines (top left, to the left of the central hexagon) and a speech bubble (bottom left).

3

Builder

Padrão de criação



Builder

Fornecer uma interface genérica para a construção incremental de agregações. Esse padrão esconde os detalhes de como os componentes são criados, representados e compostos.



Builder

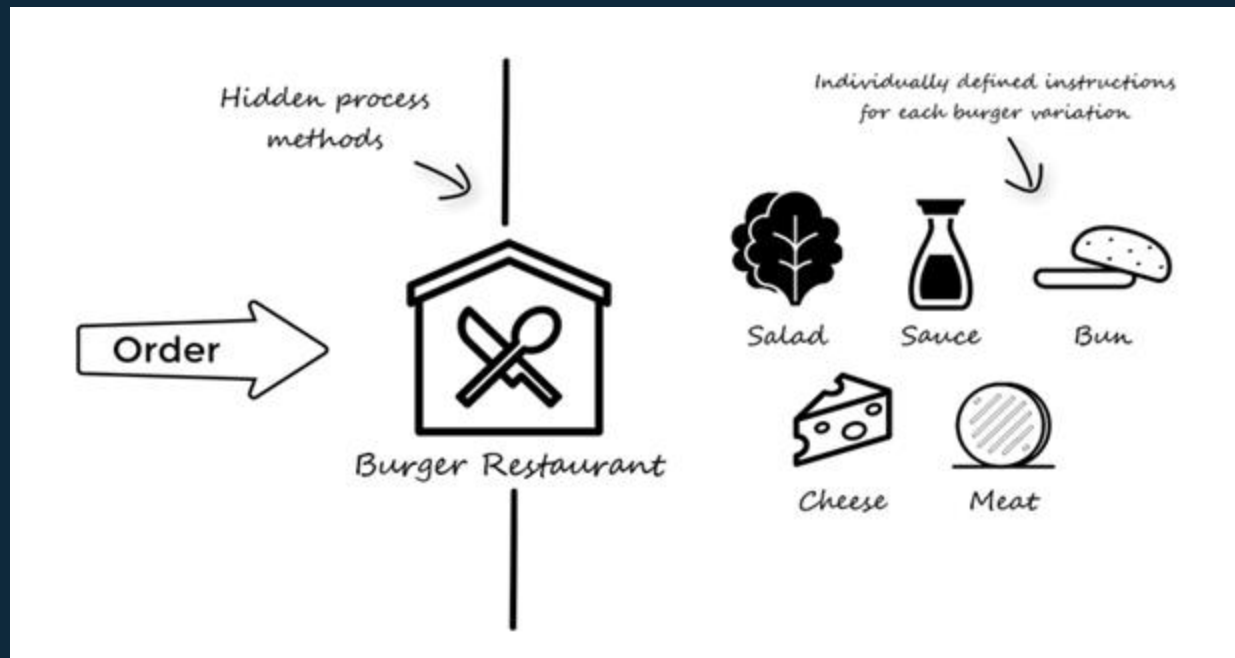


Figura 3

Builder

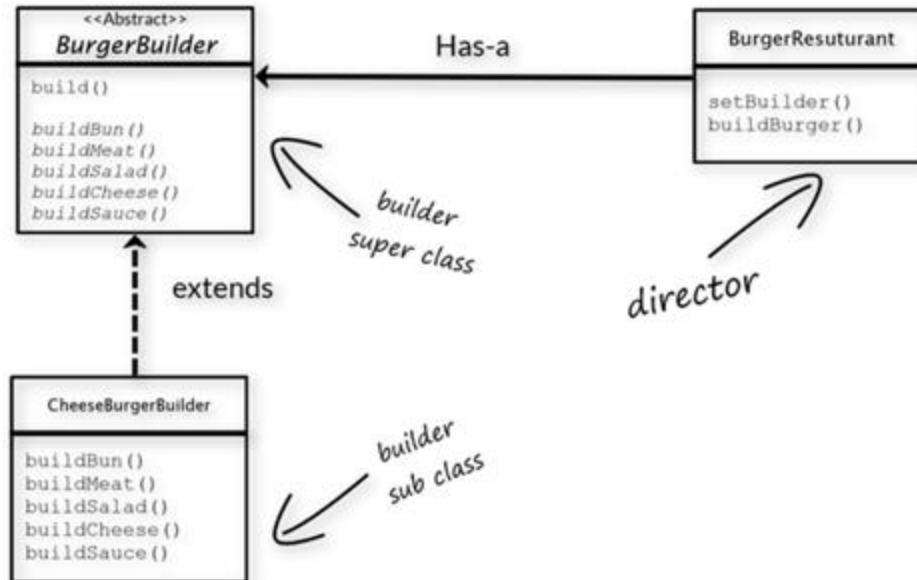


Figura 4

A decorative graphic on the left side of the slide. It features a large central hexagon with a blue-to-cyan gradient, containing the number '4'. Surrounding this central hexagon are several smaller hexagons of varying shades of blue and cyan. Some of these smaller hexagons contain white icons: a lightbulb, a thumbs-up, a smartphone, a magnifying glass, and a gear. There is also a network-like icon with a central node and several smaller nodes connected by lines.

4

Abstract Factory

Padrão de criação

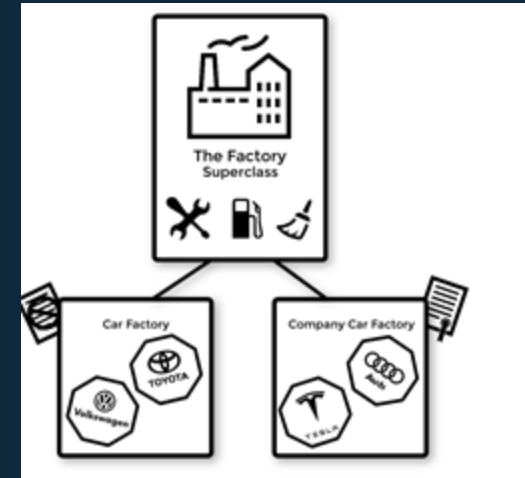
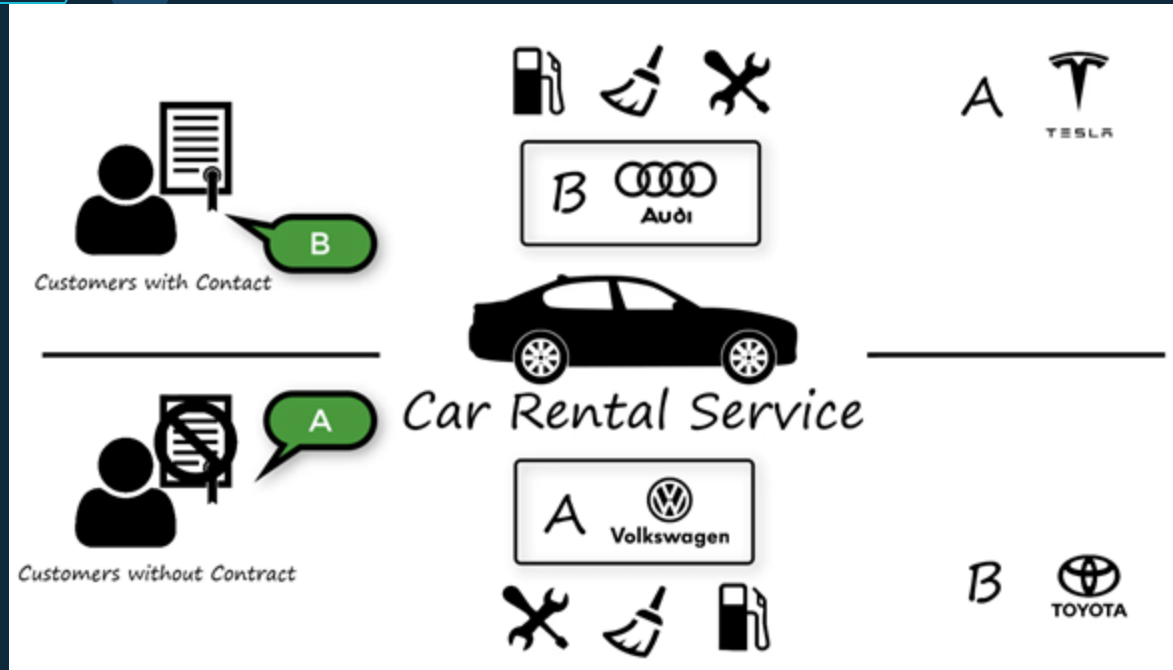


Abstract Factory

Permite elaborar uma interface para criação de famílias de objetos relacionados ou interdependentes, que não especifica suas classes concretas. A partir desse padrão consegue-se criar fábricas concretas, que são responsáveis pela criação de novos objetos para atender as necessidades do cliente. Portanto, essa prática ajuda a excluir a dependência entre o cliente e a classe dos objetos usados por ele.



Abstract Factory



Figuras 5 e 6

Abstract Factory

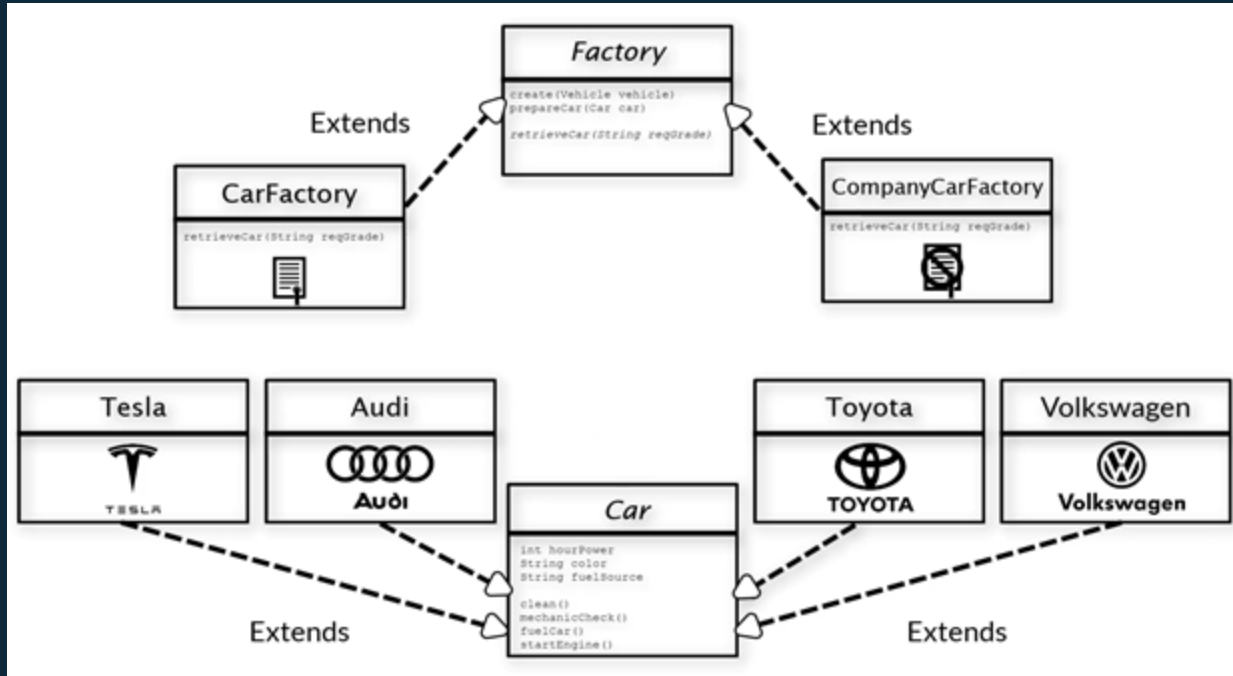


Figura 7



Referências

[Exemplos disponíveis no meu github:](#)

<https://github.com/digaomilleniun/backend-java-ebac>

<https://www.devmedia.com.br/entendendo-os-conceitos-dos-padroes-de-projetos-em-java/29083>

<http://integu.net/factory-pattern/>

<https://refactoring.guru/pt-br/design-patterns/java>

→ **Figura 1, 2, 3, 4, 5, 6 e 7:**

◆ <http://integu.net/factory-pattern/>

