

Aula 2: Cultura e identidade: origens e natureza

Apresentação

Nesta aula, iniciaremos o estudo dos padrões GoF. Enunciaremos, portanto, três de seus cinco padrões de criação: Abstract Factory, Builder e Factory Method. Debateremos as suas intenções e quais problemas eles se propõem a resolver.

Em seguida, articularemos uma apresentação resumida da estrutura do padrão, indicando outros semelhantes, o seu diagrama UML e um exemplo de sua utilização no mundo real ou em projetos de software.

Objetivos

- Descrever as características do Abstract Factory;
- Identificar os tópicos do Builder;
- Examinar os aspectos do Factory Method.

Primeiras palavras

Conforme apontamos na aula anterior, os **padrões de criação** abstraem e/ou adiam o processo de criação dos objetos, tornando o sistema independente em relação à forma como eles são criados, compostos e representados.

Padrão	Escopo Classe	Escopo Objeto
Abstract Factory		X
Builder		X
Factory Method	X	
Prototype		X
Singleton		X

Os padrões GoF Prototype e Singleton – como já frisamos – serão abordados na próxima aula. Relacionaremos agora os três primeiros da tabela acima: dois do escopo objeto (Abstract Factory e Builder) e um de classe (Factory Method).

Abstract Factory

O Abstract Factory irá produzir uma hierarquia que encapsula muitas plataformas possíveis e a construção de um conjunto de produtos, como destaca Gamma:

"Este padrão provê uma interface para construir uma família de objetos relacionados ou interdependentes que compartilham um tema em comum, sem especificar suas classes concretas."

- Gamma *et al.*, 1994

Problema

Se um aplicativo for portátil para várias plataformas, ele precisará encapsular as suas dependências, que podem incluir



`<i class="fa fa-window-restore"></i>`

Sistema de janelas



`<i class="fa fa-cogs"></i>`

Sistema operacional



`<i class="fa fa-database"></i>`

Banco de dados

Quando esse `encapsulamento`¹ não puder ser planejado antecipadamente, muitas instruções `#ifdef`, com opções para todas as plataformas suportadas atualmente, serão inseridas no código, tornando-se um problema para futuras manutenções dos programas.

Estrutura

Identifique o conjunto de produtos que precisam ser criados para diversas plataformas.



Modele cada produto como uma hierarquia de herança², fazendo a abstração³ do comportamento do produto em uma interface⁴ ou uma classe base e as realizações específicas da abstração para cada plataforma em classes derivadas.



Modele o conceito de *plataforma* com uma hierarquia de herança. Métodos como criarProdutoX e criarProdutoY, entre outros, são declarados como classes abstratas. Cada método é implementado em uma plataforma específica de classes derivadas pela chamada do construtor new() na classe derivada produto mais apropriada.



Empacote a plataforma de hierarquias para o conjunto de hierarquias de produtos.



Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Diagrama UML do padrão

O padrão Builder utiliza uma interface única compartilhada com todos os objetos que constroem outros objetos. Esta interface tem a responsabilidade de definir esse processo de construção.

Builder contém os seguintes elementos:



Director

Constrói um objeto utilizando a interface do Builder.



Builder

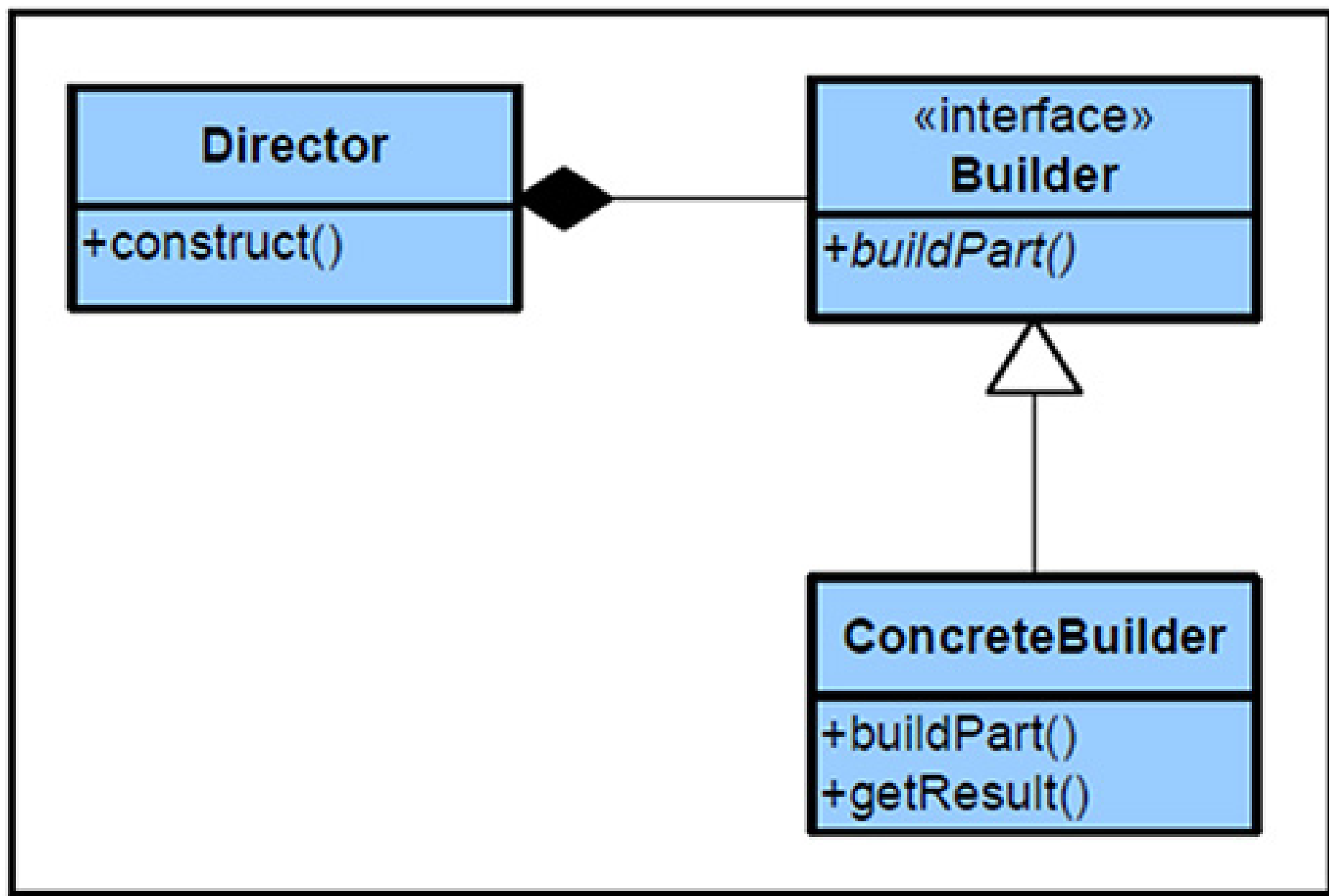
Especifica uma interface para um construtor de partes do objeto-produto.



Concrete Builder

Define uma implementação da interface Builder, mantém a representação que cria e fornece interface para recuperação do produto.


Veja o diagrama do padrão:



 (Fonte: GAMMA et al., 1994)

Agora, veremos algumas dicas e um exemplo de aplicação desse padrão. Vamos lá!

 Dicas e exemplo – padrão Builder

 Clique no botão acima.

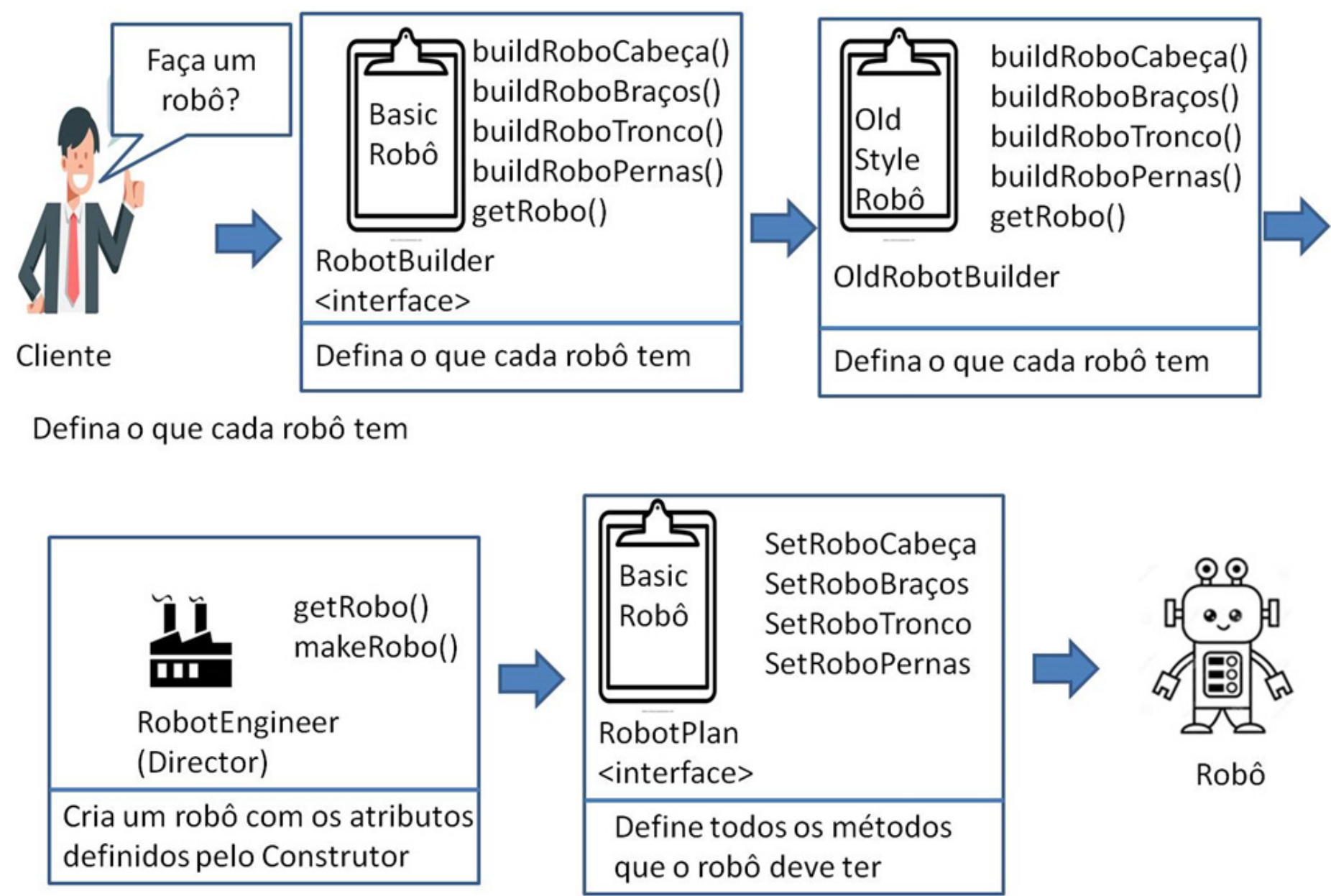
Dicas e exemplo – padrão Builder

Dicas

- Existe semelhança entre os padrões Builder e Abstract Factory, pois ambos são utilizados para construir objetos complexos. A principal diferença entre eles é que o aquele o faz passo a passo, enquanto este constrói famílias de objetos simples ou complexas de uma só vez;
- O código da construção é isolado do código da representação. Ambos são facilmente substituídos sem afetar o outro;
- Às vezes, os padrões de criação são complementares: o Builder pode usar um dos outros para implementar quais componentes serão criados;
- O Abstract Factory, o Builder e o Prototype podem usar o Singleton em suas implementações.

Exemplo

Neste exemplo, o cliente solicita a construção de um robô. A classe RobotBuilder define uma interface para os métodos para construção das partes do robô. OldRobotBuilder implementa os métodos de RobotBuilder, enquanto a classe director (RobotEngineer) cria um robô com as informações passadas pelo Builder, utilizando a interface RobotPlan para definir os valores de todos os métodos.



O padrão Builder é usado por restaurantes fast-food para construir as refeições das crianças. Elas normalmente consistem em um sanduíche, uma fruta, uma bebida e um brinquedo. Observe que pode haver uma variação no conteúdo da refeição das crianças, mas o processo de construção é o mesmo. Se um cliente pede hambúrguer, cheeseburger ou frango, ele não mudará.

O funcionário do balcão orienta a equipe a montar um item principal e outro secundário, além de um brinquedo.

Padrões relacionados:

- Bridge;
- Observer;
- State.

Atividade

1 - No padrão Builder, existe uma interface comum para todos os objetos que constroem outros objetos. Qual é o nome desta interface?

- a) Builder
- b) Director
- c) Concrete Builder
- d) Product
- e) Abstract Builder

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Factory Method (método Factory)

Segundo Gamma:

Este padrão define uma interface para criar um objeto, mas deixa as subclasses decidirem que classe instanciar. O padrão permite adiar a instanciação para subclasses.

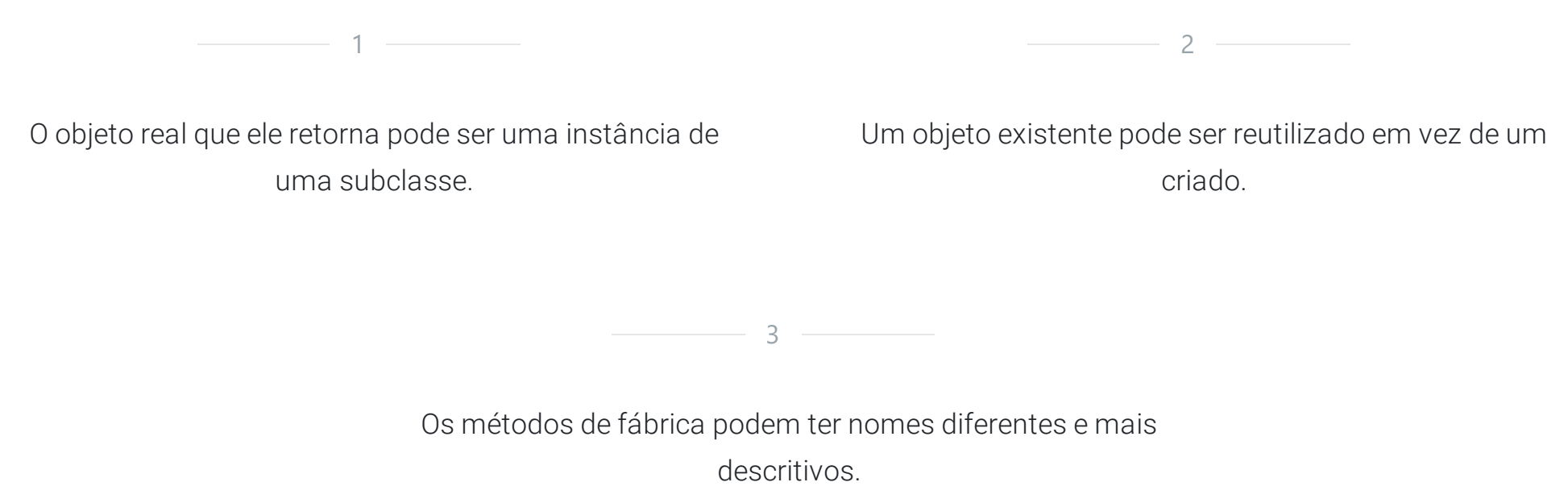
- Fonte: Gamma et al., 1994

Problema

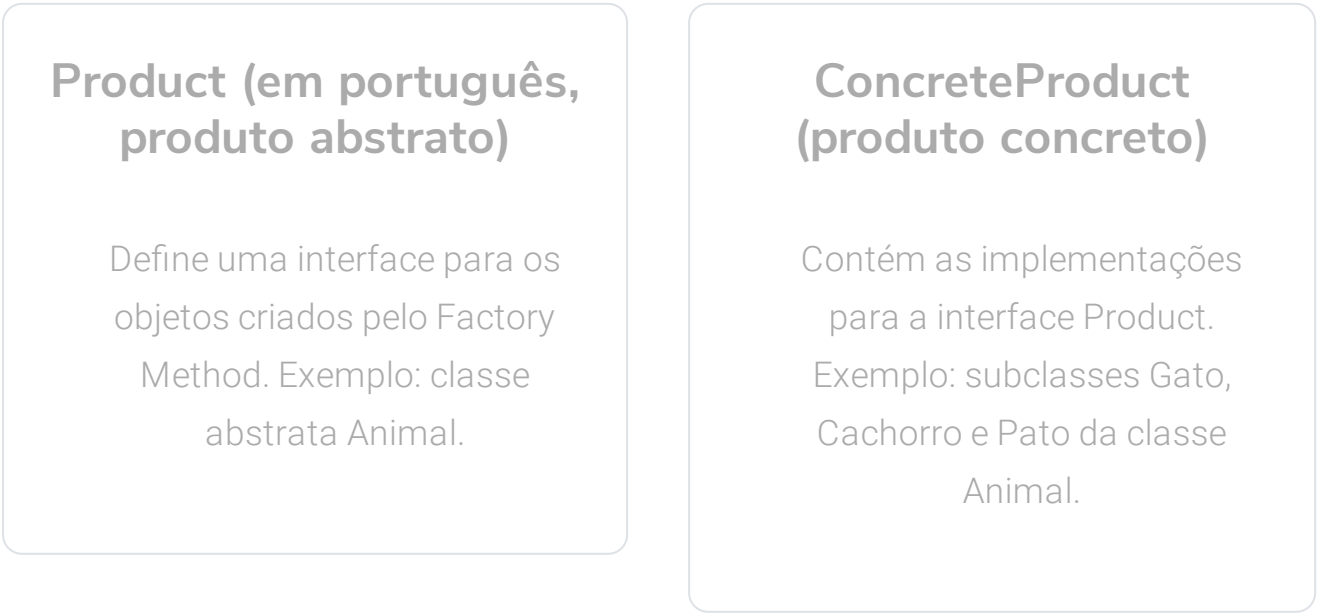
Um *framework* precisa padronizar o modelo de arquitetura para um conjunto de aplicativos, mas deve permitir que aplicativos individuais definam seus próprios objetos de domínio e forneçam sua instanciação. O acesso a um objeto concreto será por meio da interface conhecida graças à sua superclasse, embora o cliente também não queira (ou não pode) saber qual implementação concreta está usando.

Estrutura

Uma definição cada vez mais popular sobre Factory é que se trata de um método estático de uma classe que retorna um objeto desse tipo de classe. Ao contrário de um construtor:



O padrão Factory Method contém os seguintes elementos:



Creator (criador abstrato)

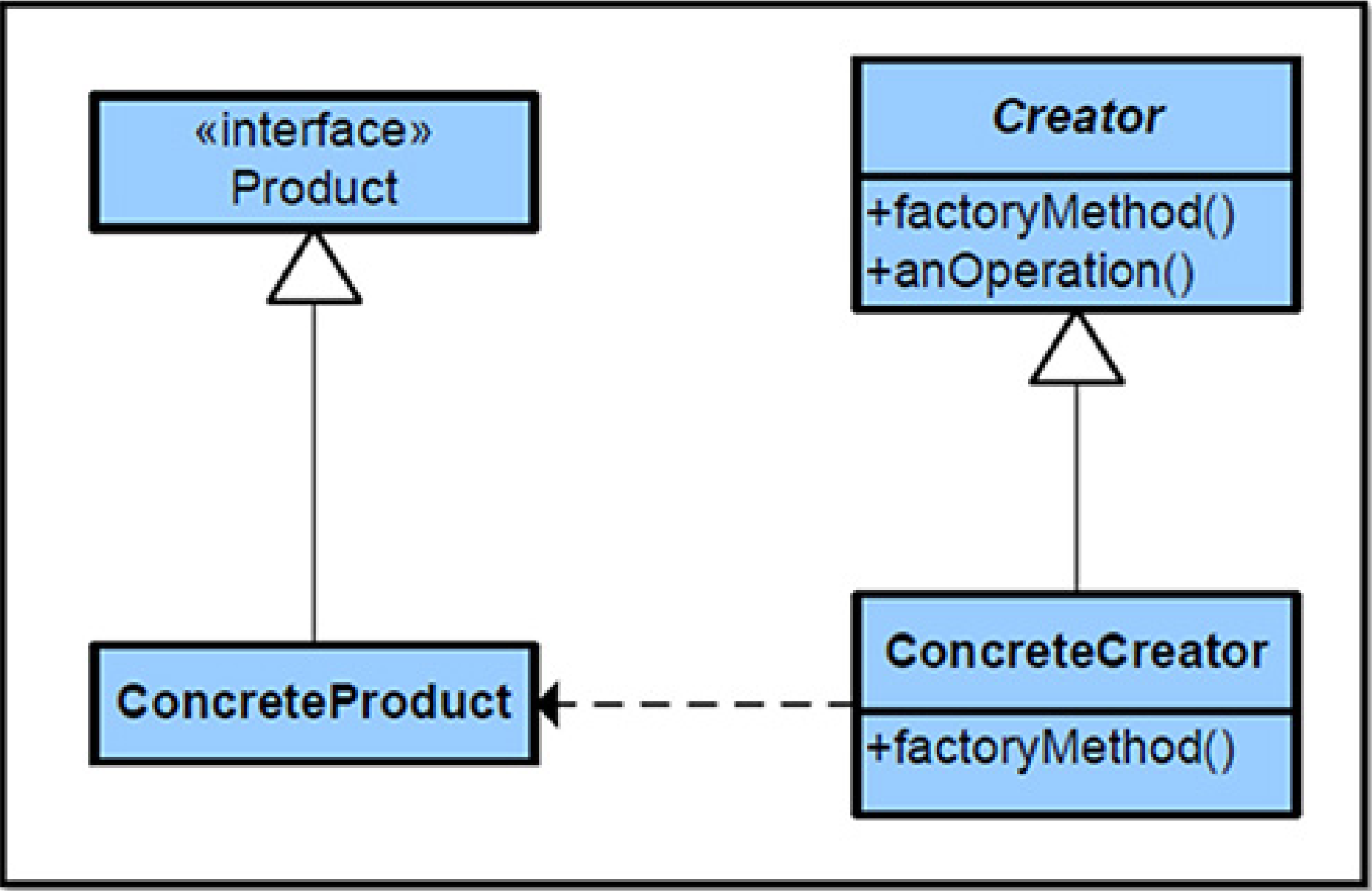
Declara o Factory Method (método de fabricação) que retorna o objeto da classe Product (produto). Exemplo: Classe abstrata Animal_Factory.

ConcreteCreator (criador concreto)

Sobrescreve o Factory Method e retorna um objeto da classe ConcreteProduct. Exemplo: Animal_Selvagem Factory, Animal_Domestico Factory.

Diagrama UML do padrão

Observe o diagrama UML do padrão Factory Method:




 (Fonte: GAMMA et al., 1994)

No padrão Factory Method, temos esta classe de criador abstrata: **Creator**, que define um método **factoryMethod()** abstrato que as subclasses implementarão para criar um produto. Ele pode ter um ou mais métodos com os seus devidos comportamentos que chamarão o **factoryMethod**. Normalmente, o **factoryMethod** do **Creator** também possui um **Product** abstrato produzido por uma subclasse (**ConcreteCreator**).

Observe que cada classe ConcreteCreator produzirá o próprio método de criação.

 Dicas e exemplo – padrão Factory Method

 Clique no botão acima.

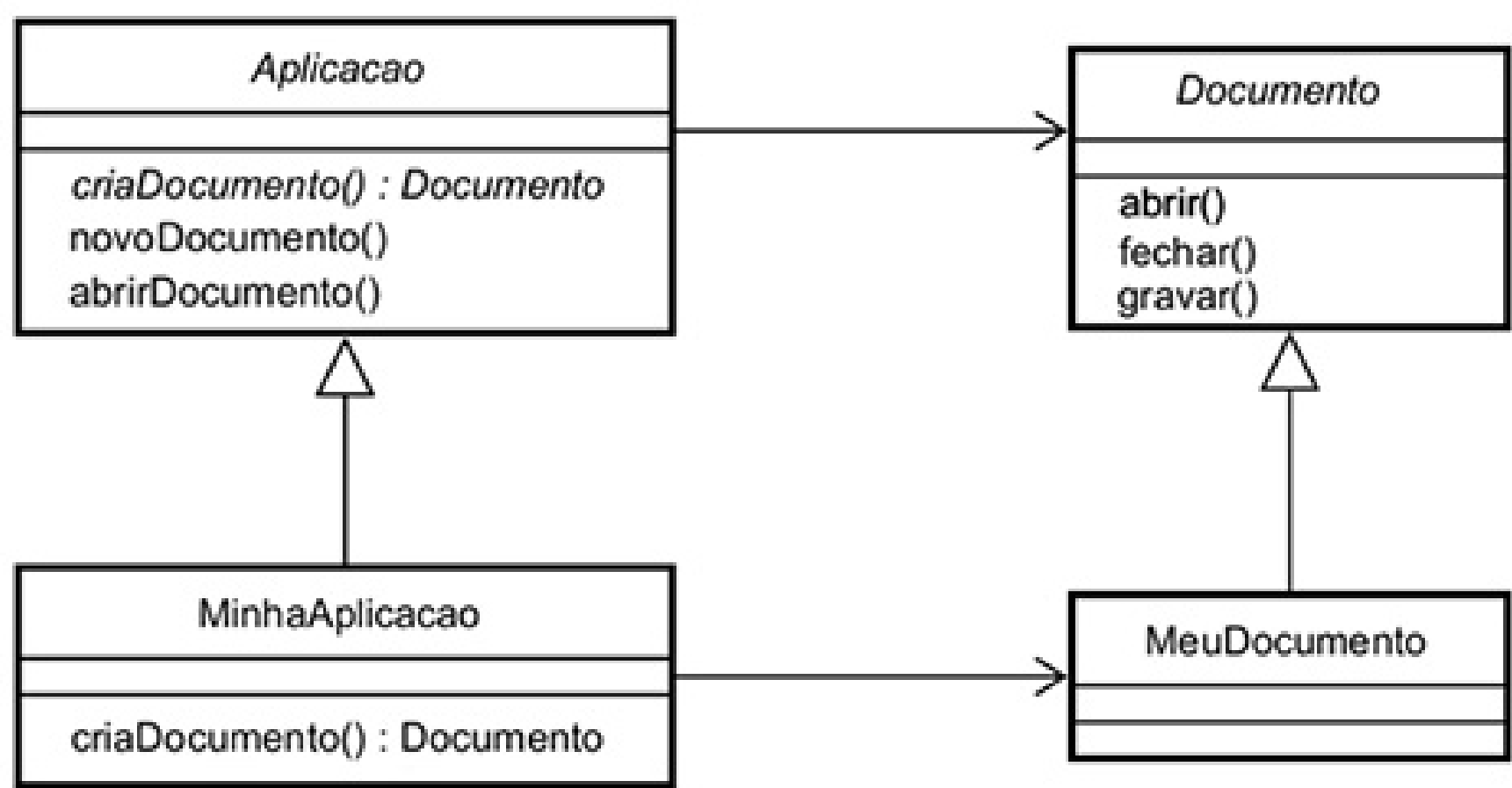
Dicas e exemplo – padrão Factory Method

Dicas

- A vantagem do método Factory é que ele pode retornar a mesma instância várias vezes ou uma subclasse em vez de um objeto desse tipo exato;
- Muitas vezes, os projetos começam usando o Factory Method (menos complicado e mais personalizável, no qual as subclasses proliferam) até evoluírem para Abstract Factory, Prototype ou Builder (mais flexível e mais complexo) à medida que o designer descobrir em qual ponto é necessário haver mais flexibilidade.

Exemplo

Este exemplo pode suportar a criação de diversos documentos. O modelo é constituído por dois tipos de classes: abstratas (*Aplicacao* e *Documento*) e concretas (*MinhaAplicacao* e *MeuDocumento*). *MinhaAplicacao* é uma implementação da abstração definida pela classe *Aplicacao*.



Observe a declaração do método abstrato *criaDocumento()* e da classe *Aplicação*, além da forma de sua utilização pelo método *novoDocumento()*. Esse arranjo permite que *novoDocumento()* crie documentos sem conhecer os detalhes de implementação existentes em cada tipo de documento suportado pela aplicação. Será possível, assim, implementar diversos formatos de documentos alterando apenas a classe *MinhaAplicação*, sem haver a necessidade de modificar o código das classes abstratas.

Padrões relacionados:

- Strategy;
- Visitor.

2 - Uma equipe de projeto está desenvolvendo um aplicativo para smartfone com um amplo conjunto de funções que deve funcionar nas plataformas IOS e Android. Qual o padrão de projeto mais adequado para ser utilizado?

- a) O padrão Builder, porque ele facilita a construção de sistemas complexos.
- b) O padrão Abstract Factory, porque ele produz uma hierarquia de classes que encapsula muitas plataformas.
- c) O padrão Factory Method, porque ele padroniza o modelo de arquitetura para um conjunto de aplicativos diferentes.
- d) O padrão Abstract Factory, porque ele utiliza muitas instruções #ifdef para várias plataformas de hardware e software.
- e) O padrão Factory Method, porque ele permite que aplicativos individuais definam os próprios objetos de domínio e forneçam sua instanciação.

3 - Qual é o relacionamento entre os padrões Abstract Factory e Factory Method?

- a) O padrão Abstract Factory é mais simples que o Factory Method.
- b) A desvantagem do Factory Method em relação ao Abstract Factory é que ele pode retornar a mesma instância várias vezes.
- c) Factory Method atua sobre o escopo de classes, enquanto Abstract Factory o faz sobre o de classes.
- d) As classes Abstract Factory podem ser implementadas com Factory Methods.
- e) As classes Factory Methods podem ser implementadas com Abstract Factory.

Notas

Encapsulamento¹

Seu conceito consiste em esconder os atributos da classe de quem for a utilizar.

Herança²

Este mecanismo permite que você baseie uma nova classe na definição daquela previamente existente. Usando a herança, ela irá herdar todos os atributos e comportamentos da classe previamente existente.

Abstração³

Processo para simplificar um problema difícil. A abstração tem duas vantagens: permite que ele seja resolvido facilmente e ajuda a obter a reutilização de classes e objetos. Utilizando-a, o código escrito pode ser reutilizado em diversas situações.

Interface⁴

Interface (ou protocolo) lista os serviços oferecidos por um componente. Ela é um contrato com o mundo exterior, que define o que uma entidade externa pode fazer com o objeto.

Referências

GAMMA, E. *et al.* **Design patterns**: elements of reusable object-oriented software. Addison-Wesley, 1994.

FRIEMAN, E. **Use a cabeça!** Padrões de projeto. 2. ed. Rio de Janeiro: Elsevier, 2007.

LEÃO, L. **Padrões de projeto de software**. Rio de Janeiro: SESES, 2018.

SINTES, A. **Aprenda programação orientada a objetos em 21 dias**. São Paulo: Makron Books, 2002.

Próxima aula

- Apresentação dos padrões Prototype, Singleton e Bridge

Explore mais

Leia o texto: [Construindo objetos de forma inteligente: Builder Pattern e Fluent Interfaces](#).

Código C# para os padrões:

- [Abstract Factory](#);
- [Builder](#);
- [Factory Method](#).

Assista ao vídeo: [Uso do padrão de projeto Abstract Factory](#).