CENTRO UNIVERSITÁRIO
FRANCISCANO
educação virtual

DESIGN PATTERNS - Prof. Fernando Prass

Padrões de Criação

Padrões de Criação (Creational)

- Padrões de criação abstraem o processo de instanciação, ajudando a tornar o sistema independente de como os objetos são criados, compostos e representados
- São padrões de criação:
  - Abstract Factory, Builder, Factory Method, Prototype e Singleton

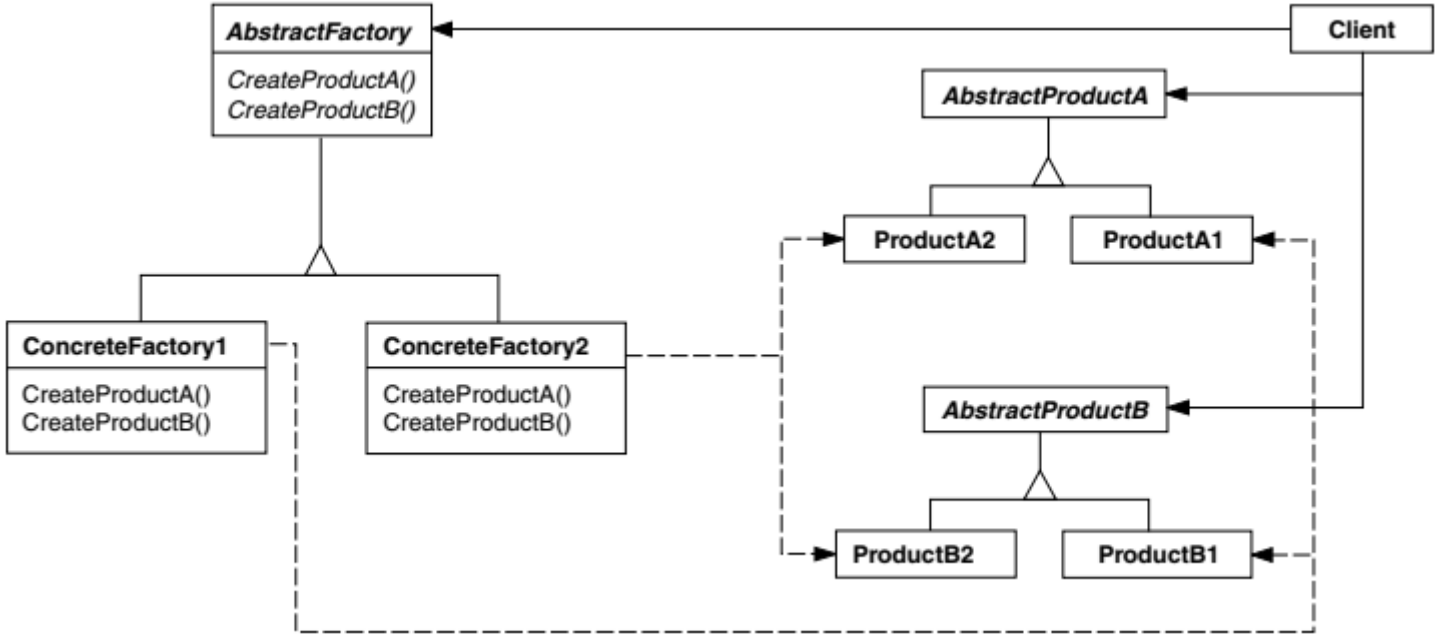
Abstract Factory

- **Objetivo**: também conhecido como “Kit”, prove uma interface para criar famílias de objetos relacionados ou dependentes sem especificar suas classes concretas.
- **Problema:** criar uma família de objetos relacionados sem conhecer suas classes concretas.
- **Padrões Relacionados:** AbstractFactory costumam ser implementadas com *Factory Method* e/ou *Prototype*. Fábricas concreta normalmente são um *Singleton*.

Abstract Factory - Estrutura

- **AbstractFactory:** declara uma interface para operações que criam objetos-produto abstratos.
- **Concrete Factory:** implementa as operações que criam objetos-produto concretos.
- **AbstractProduct:** declara uma interface para um tipo de objeto-produto
- **ConcreteProduct:** define um objeto-produto a ser criado pela correspondente fábrica concreta e implementa a interface de AbstractProduct.
- **Client: c**ria famílias de objeto-produto

Abstract Factory - Estrutura



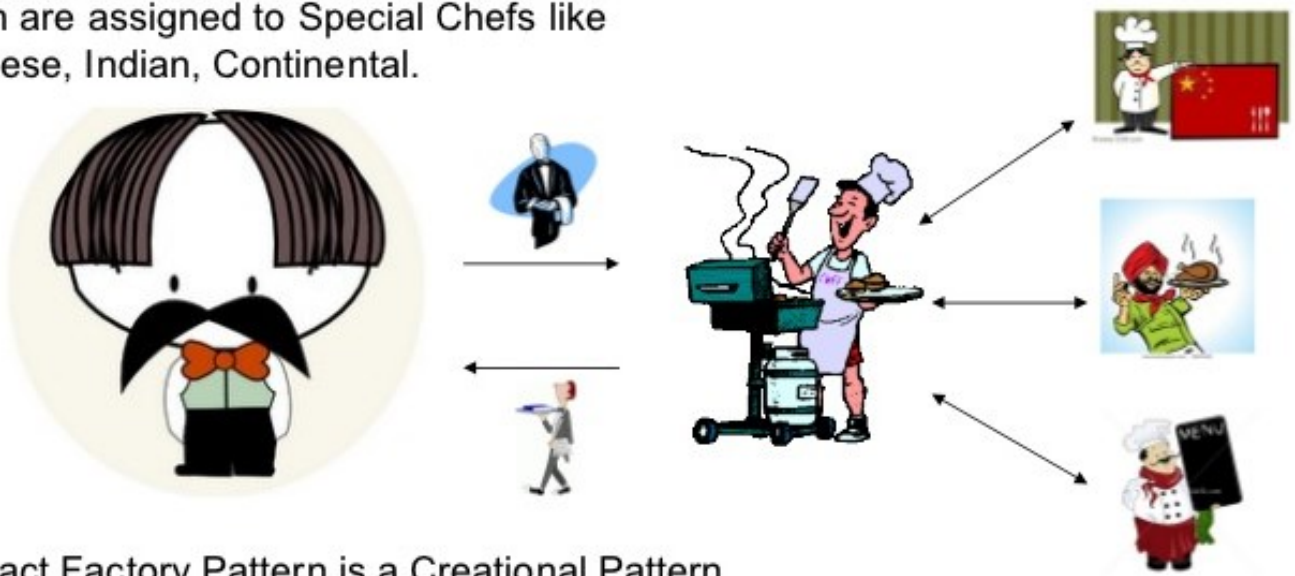
Fonte: GAMMA E. et al. **Padrões de Projeto Soluções Reutilizáveis de Software Orientado a Objetos**. Bookman, 2005.

Abstract Factory - In Real Life

Abstract Factory Pattern is similar to Sub Contracting in real world.
Basically delegating the creation of Objects to expert Factories

---

Orders in a restaurant are received by a Kitchen.
Then are assigned to Special Chefs like
Chinese, Indian, Continental.



Abstract Factory Pattern is a Creational Pattern.
Similar to Factory Pattern it is Object Creation without exposing “HOW” ?

Abstract Factory - Consequências

- **Isola as classes concretas:** ajuda a controlar as classes criadas, uma vez que encapsula a responsabilidade e o processo de criar produto, e isola os clientes das classes de implementação.
- **Torna fácil a troca de famílias de produtos:** a classe de uma fábrica concreta aparece apenas uma vez numa aplicação, o que torna fácil mudar a fábrica concreta que uma aplicação usa.
- **Promove a harmonia entre produtos:** quando produtos família são projetados para trabalharem juntos, é importante que uma aplicação use objetos de somente uma família de cada vez.
- **É difícil de suportar novos tipos de produtos:** suportar novos tipos de produto exige estender a interface da fábrica, o que envolve mudar a classe AbstractFactory e todas as suas subclasses.

Abstract Factory - Hands on

- Let’s assume there is a car manufacturing factory and it manufactures different segments of the cars. It produces mini and compact Sedans and also SUV’s. The factory must be having separate units of manufacturing these vehicles.
- The problem occurs when the manufacturing company doesn’t want to manufacture compact and full cars together. So here Abstract Factory comes into the picture.
- Manufacturing units should not directly manufacture the cars, it should prepare and instantiate the car through a factory and this factory tells them that what type of car needs to be manufactured.
    - Fonte: www.codeproject.com/Articles/1252464/Abstract-Factory-Design-Pattern-with-simple-Csharp

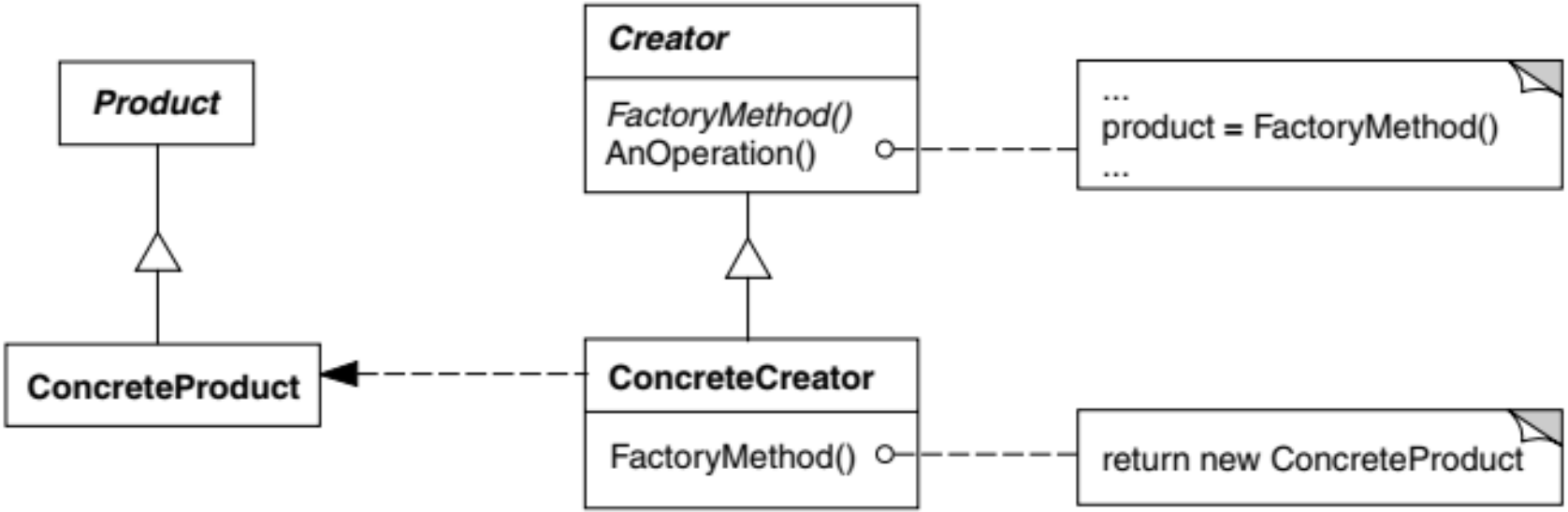
Factory Method

- **Objetivo**: Também conhecido como “Virtual Constructor”, define uma interface para criar um objeto mas deixar que subclasses decidam que classe instanciar. Factory Method permite que uma classe delegue a responsabilidade de instanciação às subclasses.
- **Padrões Relacionados:** *Abstract Factory* é frequentemente implementado utilizando o padrão, além disto, são usualmente chamados dentro de *Template Methods*. *Prototypes* não exigem subclassificação de *Creator*, contudo, frequentemente necessitam uma operação Initialize na classe Product. A Creator usa *Initialize* para iniciar o objeto, o *Factory Method* não exige uma operação desse tipo.

Factory Method – Quando Usar

- Uma classe não pode antecipar a classe de objetos que deve criam;
- Uma classe quer que suas subclasses especifiquem os objetos que criam;
- Classes delegam responsabilidade para uma dentre várias subclasses auxiliares, e você quer localizar o conhecimento de qual subclasse auxiliar que é a delegada.

Factory Factory - Estrutura



Fonte: GAMMA E. et al. **Padrões de Projeto Soluções Reutilizáveis de Software Orientado a Objetos**. Bookman, 2005.

Factory Method - Estrutura

- **Product:** define a interface de objetos que o método fábrica cria.
- **ConcreteProduct:** implementa a interface de Product.
- **Creator:** Declara o método fábrica, o qual retorna um objeto do tipo Product.
- **Creator:** pode também definir uma implementação por omissão do método factory que retorna por omissão um objeto ConcreteProduct. Pode chamar o método factory para criar um objeto Product.
- **ConcreteCreator:** Redefine o método-fábrica para retornar a uma instância de um ConcreteProduct

Factory Method - Consequências

- **Elimina a necessidade de anexar classes específicas das aplicações no código:** o código lida somente com a interface de Product, podendo trabalhar com quaisquer classes ConcreteProduct definidas pelo usuário.
- **Fornece subclasses da classe Creator:** somente para criar um objeto ConcreteProduct em particular. Usar subclasses é bom quando o cliente tem que fornecer subclasses a Creator de qualquer maneira, caso contrário, o cliente deve lidar com outro ponto de evolução.

Abstract Factory - Hands on

- Let’s assume there is a car manufacturing factory and it manufactures different segments of the cars. It produces mini and compact Sedans and also SUV’s. The factory must be having separate units of manufacturing these vehicles.
- The problem occurs when the manufacturing company doesn’t want to manufacture compact and full cars together. So here Abstract Factory comes into the picture.
- Manufacturing units should not directly manufacture the cars, it should prepare and instantiate the car through a factory and this factory tells them that what type of car needs to be manufactured.
    - Fonte: www.codeproject.com/Articles/1252464/Abstract-Factory-Design-Pattern-with-simple-Csharp

Factory Method vs Abstract Factory

- Factory Method é usado para criar um produto somente e Abstract Factory é sobre a criação de famílias de produtos relacionados ou dependentes (talvez a diferença mais simples de ser entendida - e certamente a mais importante)
- Factory Method depende de herança para decidir qual produto a ser criado (esta é diferença mais confusa, pois ambos parecem estar usando herança)

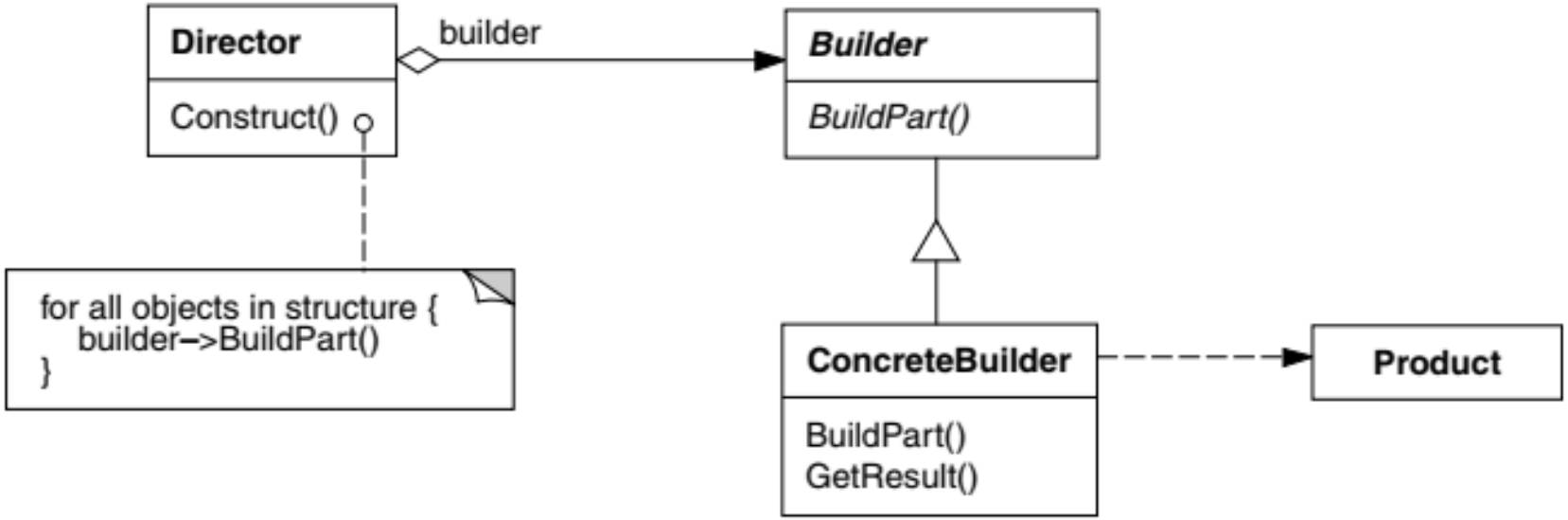
Factory Method vs Abstract Factory

- Factory Method é apenas um método, enquanto Abstract Factory é um objeto.
- Abstract Factory possui um nível mais alto na abstração do que Factory Method . Enquanto Factory Method abstrai a forma como os objetos são criados, Abstract Factory abstrai a forma como as fábricas são criadas, que por sua vez abstrai a forma como os objetos são criados.
- Como Abstract Factory está em nível mais alto na abstração, ele geralmente usa o Factory Method para criar os produtos em fábricas.

Builder

- **Objetivo:** Separa a construção de um objeto complexo da sua representação de modo que o mesmo processo de construção possa criar diferentes representações.
- **Padrões relacionados**: semelhante ao *Abstract Factory,* no sentido de que também pode construir objetos complexos. A diferença principal é que o Builder focaliza a construção de um objeto complexo e a ênfase do Abstract Factory é sobre famílias de objetos-produto. Um *Composite* é o que frequentemente o builder constrói.

Builder – Estrutura



Fonte: GAMMA E. et al. **Padrões de Projeto Soluções Reutilizáveis de Software Orientado a Objetos**. Bookman, 2005.

Builder - Estrutura

- **Builder:** especifica uma interface abstrata para criação de partes de um objeto-produto.
- **ConcreteBuilder:** constrói e monta partes do produto pela implementação da interface de Builder;
  - define e mantém a representação que cria;
  - fornece uma interface para recuperação do produto
- **Director**: constrói um objeto usando a interface de Builder.
- **Product**: representa o objeto complexo em construção. ConcreteBuilder constrói a representação interna do produto e define o processo pelo qual ele é montado; inclui classes que definem as partes constituintes, inclusive as interfaces para a montagem das partes no resultado final.

Builder – Quando Usar

- Builder permite que uma classe se preocupe com apenas uma parte da construção de um objeto. É útil em algoritmos de construção complexos, use quando o algoritmo para criar um objeto complexo precisar ser independente das partes que compõem o objeto e da forma como o objeto é construído;
- Builder também suporta substituição dos construtores, permitindo que a mesma interface seja usada para construir representações diferentes dos mesmos dados. Use quando o processo de construção precisar suportar representações diferentes do objeto que está sendo construído

Builder - Consequências

- **Permite variar a representação interna de um produto:** fornece ao diretor uma interface abstrata para a construção do produto. A interface permite ao construtor ocultar a representação e a estrutura interna do produto, além de ocultar como o produto é montado.
- **Isola o código para construção e representação:** o padrão melhora a modularidade pelo encapsulamento da forma como um objeto complexo é construído e representado. Os clientes nada necessitam saber sobre as classes que definem a estrutura interna do produto; tais classes não aparecem na interface de Builder.
- **Oferece um controle mais fino sobre o processo de construção:** ao contrário de padrões de criação que constroem produtos de uma só vez, o Builder constrói o produto passo a passo sob o controle do diretor.

Builder - Exemplo Prático

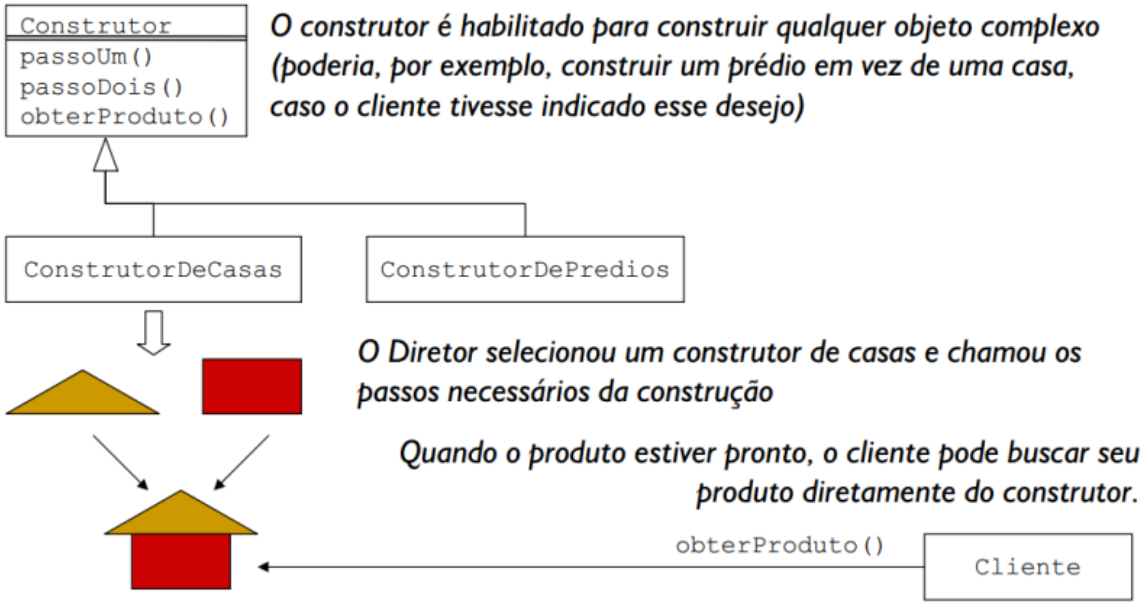
Cliente

Cliente precisa de uma casa. Passa as informações necessárias para seu diretor

Problema

Diretor

Utilizando as informações passadas pelo cliente, ordena a criação da casa pelo construtor usando uma interface uniforme

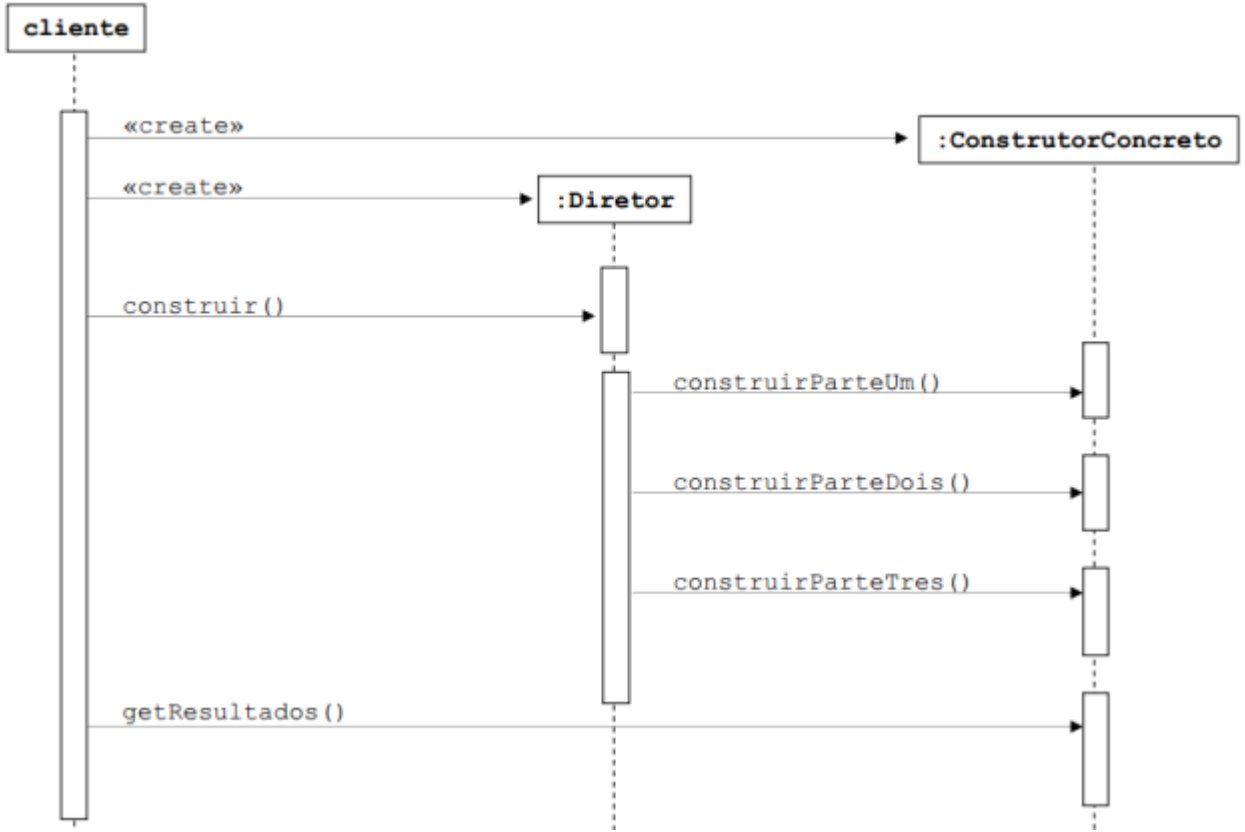


O construtor é habilitado para construir qualquer objeto complexo (poderia, por exemplo, construir um prédio em vez de uma casa, caso o cliente tivesse indicado esse desejo)

O Diretor selecionou um construtor de casas e chamou os passos necessários da construção

Quando o produto estiver pronto, o cliente pode buscar seu produto diretamente do construtor.

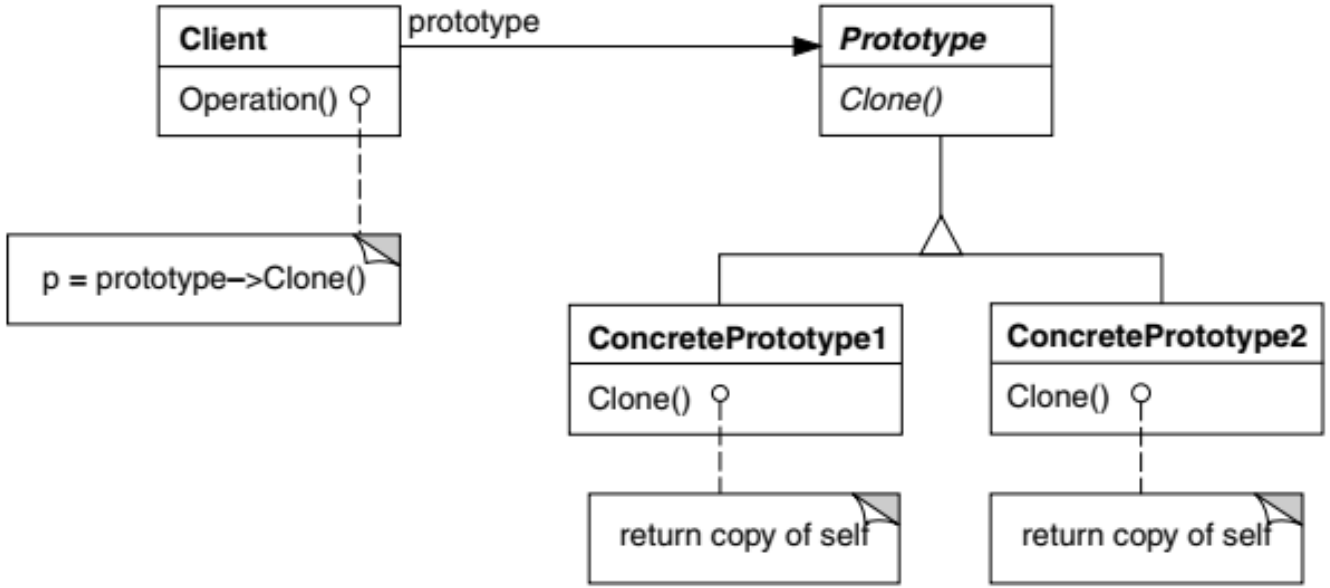
Builder - Exemplo Prático



Prototype

- **Objetivo:** Especificar os tipos de objetos a serem criados usando uma instância-protótipo e criar novos objetos pela cópia desse protótipo.
- **Padrões relacionados:** Prototype e Abstract Factory são padrões que competem entre si em várias situações. Porém, eles também podem ser usados em conjunto. Um Abstract Factory pode armazenar um conjunto de protótipos a partir dos quais podem ser clonados e retornados objetos-produto.

Prototype – Estrutura



Fonte: GAMMA E. et al. **Padrões de Projeto Soluções Reutilizáveis de Software Orientado a Objetos**. Bookman, 2005.

Prototype - Estrutura

- **Prototype**: declara uma interface para clonar a si próprio.
- **ConcretePrototype**: implementa uma operação para clonar a si próprio.
- **Client**: cria um novo objeto solicitando a um protótipo que clone a si próprio

Prototype – Quando Usar

- Quando as classes a instanciar forem especificadas em tempo de execução, por exemplo, por carga dinâmica
- Para evitar a construção de uma hierarquia de classes de fábricas paralela à hierarquia de classes de produto
- Quando as instâncias de uma classe puderem ter uma dentre poucas combinações diferentes de estados. Pode ser mais conveniente instalar um número correspondente de protótipos e cloná-los, ao invés de instanciar a classe manualmente, cada vez com um estado apropriado.

Prototype - Consequências

- **Acrescenta e remove produtos em tempo de execução:** permite incorporar uma nova classe concreta de produto a um sistema, simplesmente registrando uma instância protótipo com o cliente. Isso é um pouco mais flexível do que outros padrões de criação, porque o cliente pode instalar e remover protótipos em tempo de execução.
- **Especifica novos objetos pela variação de valores:** sistemas altamente dinâmicos permitem definir novos comportamentos através da composição de objetos
  - por exemplo, pela especificação de valores para as variáveis de um objeto
  - e não pela definição de novas classes.

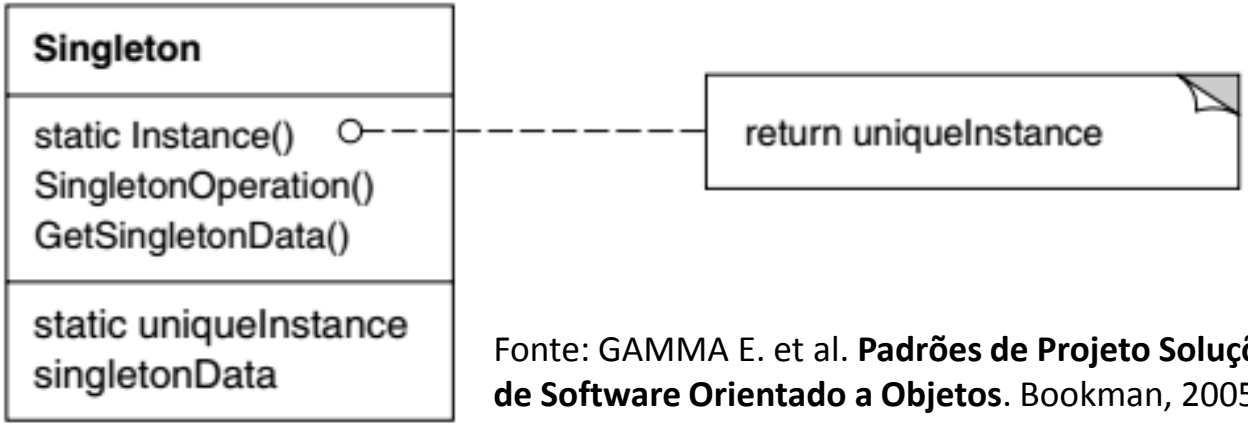
Prototype - Consequências

- **Especifica novos objetos pela variação da estrutura:** muitas aplicações constroem objetos com partes e subpartes. Por questões de conveniência, tais aplicações frequentemente permitem instanciar estruturas complexas, definidas pelo usuário.
- **Reduz o número de subclasses:** Factory Method normalmente produz uma hierarquia de classes Creator paralela à hierarquia do produto. Prototype permite clonar um protótipo em vez de pedir a um método fábrica para construir um novo objeto.
- **Configura dinamicamente uma aplicação com classes:** alguns ambientes de tempo de execução permitem carregar classes dinamicamente numa aplicação.

Singleton

- **Objetivo:** garantir que uma classe tenha somente uma instância e fornecer um ponto global de acesso para a mesma.
-

Singleton – Estrutura



Fonte: GAMMA E. et al. **Padrões de Projeto Soluções Reutilizáveis de Software Orientado a Objetos**. Bookman, 2005.

- **Singleton**: define uma operação Instance que permite aos clientes acessarem sua única instância. Pode ser responsável pela criação da sua própria instância única.

Singleton – Quando Usar

- Quanto for preciso haver apenas uma instância de uma classe, e essa instância tiver que dar acesso aos clientes através de um ponto bem conhecido
- Quando a única instância tiver de ser extensível através de subclasses, possibilitando aos clientes usar uma instância estendida sem alterar o seu código.

Singleton - Consequências

- **Acesso controlado à instância única:** como a classe Singleton encapsula a sua única instância, possui controle total sobre como e quando os clientes a acessam.
- **Espaço de nomes reduzido:** O padrão Singleton representa uma melhoria em relação ao uso de variáveis globais. Ele evita a poluição do espaço de nomes com variáveis globais que armazenam instâncias únicas.
- **Permite um refinamento de operações e da representação:** a classe Singleton pode ter subclasses e é fácil configurar uma aplicação com uma instância dessa classe estendida. Pode-se configurar a aplicação com uma instância da classe de que necessita em tempo de execução.

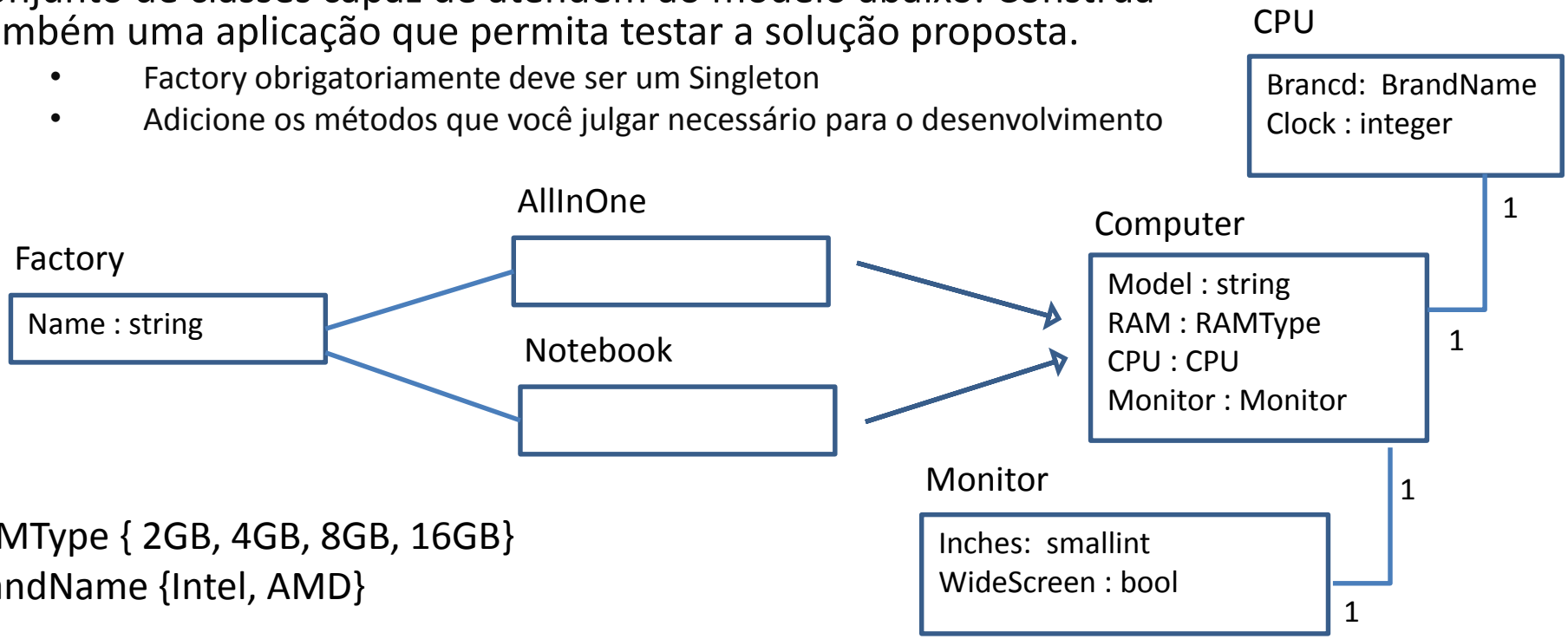
Singleton - Consequências

- **Permite um número variável de instâncias:** o padrão torna fácil mudar de ideia, permitindo mais de uma instância da classe Singleton. Além disso, pode-se usar a mesma abordagem para controlar o número de instâncias que a aplicação utiliza. Somente a operação que permite acesso à instância de Singleton necessita ser mudada.
- **Mais flexível do que operações de classe:** outra maneira de empacotar a funcionalidade de um Singleton é usando operações de classe , entretanto as técnicas das linguagens de programação tornam difícil mudar um projeto para permitir mais que uma instância de uma classe.

Trabalho Prático

Usando ao menos três dos cincos padrões criacionais, implemente o conjunto de classes capaz de atendem ao modelo abaixo. Construa também uma aplicação que permita testar a solução proposta.

- Factory obrigatoriamente deve ser um Singleton
- Adicione os métodos que você julgar necessário para o desenvolvimento



RAMType { 2GB, 4GB, 8GB, 16GB}
BrandName {Intel, AMD}

Bibliografia

- BOOCH, Grady. RUMBAUGH, James. JACOBSON, Ivar. **The Unified Modeling Language User Guide.** J. Database Manag, 1999.
- GAMMA E. et al. **Padrões de Projeto Soluções Reutilizáveis de Software Orientado a Objetos**. Bookman, 2005.
- METSKER, Steven John. **Design Patterns in C#,** Addison-Wesley Professional, 2004.

CENTRO UNIVERSITÁRIO
FRANCISCANO
educação virtual