Programação Orientada a Objetos II

Aula 03

Prof. Leandro Nogueira Couto
UFU – Monte Carmelo

Patterns

- Um padrão de é uma maneira de documentar uma solução conhecida para um problema usualmente encontrado
- O objetivo do padrão é permitir que boas soluções sejam reutilizadas em diferentes projetos
- Um padrão de projeto possui 3 partes distintas
  - **Contexto**
  - **Problema recorrente** neste contexto
  - **Solução** para o problema
-

Patterns

- Design Patterns
  - Uma nova categoria de conhecimento
  - O conhecimento não é novo, mas falar sobre ele é
  - Te tornam um melhor designer
  - Melhora comunicação entre projetistas

Patterns

- Padrões não são idiomas ou linguagens
- Padrões não são algoritmos
- Padrões não são componentes
- Padrões não são “silver bullet” (solução para todos os problemas)

Patterns

- Características dos Padrões de Projeto
  - São observados através da **experiência**
  - São descritos de uma forma **estruturada**
  - Previnem **contra a "reinvenção da roda"**
  - Existem em diferentes níveis de abstração
  - Estão **em desenvolvimento** contínuo
  - São **artefatos** reutilizáveis
  - Transmitem melhores **práticas**
  - Permitem o uso de um **vocabulário comum**
  - Podem ser utilizados **em conjunto** para resolver um problema mais amplo

Patterns

- Os padrões de projeto podem ser classificados de acordo com a fase de desenvolvimento em que são mais adequados:
- **Padrões de Análise** (Analysis patterns)
  - Seu foco é na fase de análise ou modelamento de negócio
  - Padrões ligados ao domínio do problema
- **Padrões de Arquitetura** (Architectural patterns)
  - Seu foco é na arquitetura do software
- **Padrões de Projeto** (Design patterns)
  - Foco no projeto de componentes do software
- Muitas das vezes os padrões podem estar muito ligados tanto ao **domínio** da solução, quanto do **problema**

Patterns

- Padrões de Análise (Analysis patterns)
  - Martin Fowler , 1996
- Padrões de Arquitetura (Architectural patterns)
  - Apresentado inicialmente por Frank Buschmann et al., 1996
  - Computação Distribuída - Frank Buschmann et al., 2007
- Padrões de Projeto (Design patterns)
  - GOF (Gang of Four) E. Gamma, R. Helm, R. Johnson, J. Vlissides – 1995
  - Aplicações Concorrentes e em Rede - Frank Buschmann et al. – 2000
  - Enterprise Integration Patterns – Gregor Hohpe, 2003
  - Real-time Design Patterns – Bruce Douglass, 2003
  - .Net Design Patterns - Christian Thilmany, 2003
  - J2EE Design Patterns - Deepak Alur, 2003
  - Web Services Patterns – Paul Monday, 2003
  - Ajax Design Patterns - Michael Mahemoff, 2006
  - SOA Design Patterns – Thomas Erl, 2009

A Inspiração

- A idéia de padrões foi apresentada por Christopher Alexander em 1977 no contexto de Arquitetura (de prédios e cidades):
-

  Cada padrão descreve um problema que ocorre repetidamente de novo e de novo em nosso ambiente, e então descreve a parte central da solução para aquele problema de uma forma que você pode usar esta solução um milhão de vezes, sem nunca implementa-la duas vezes da mesma forma.

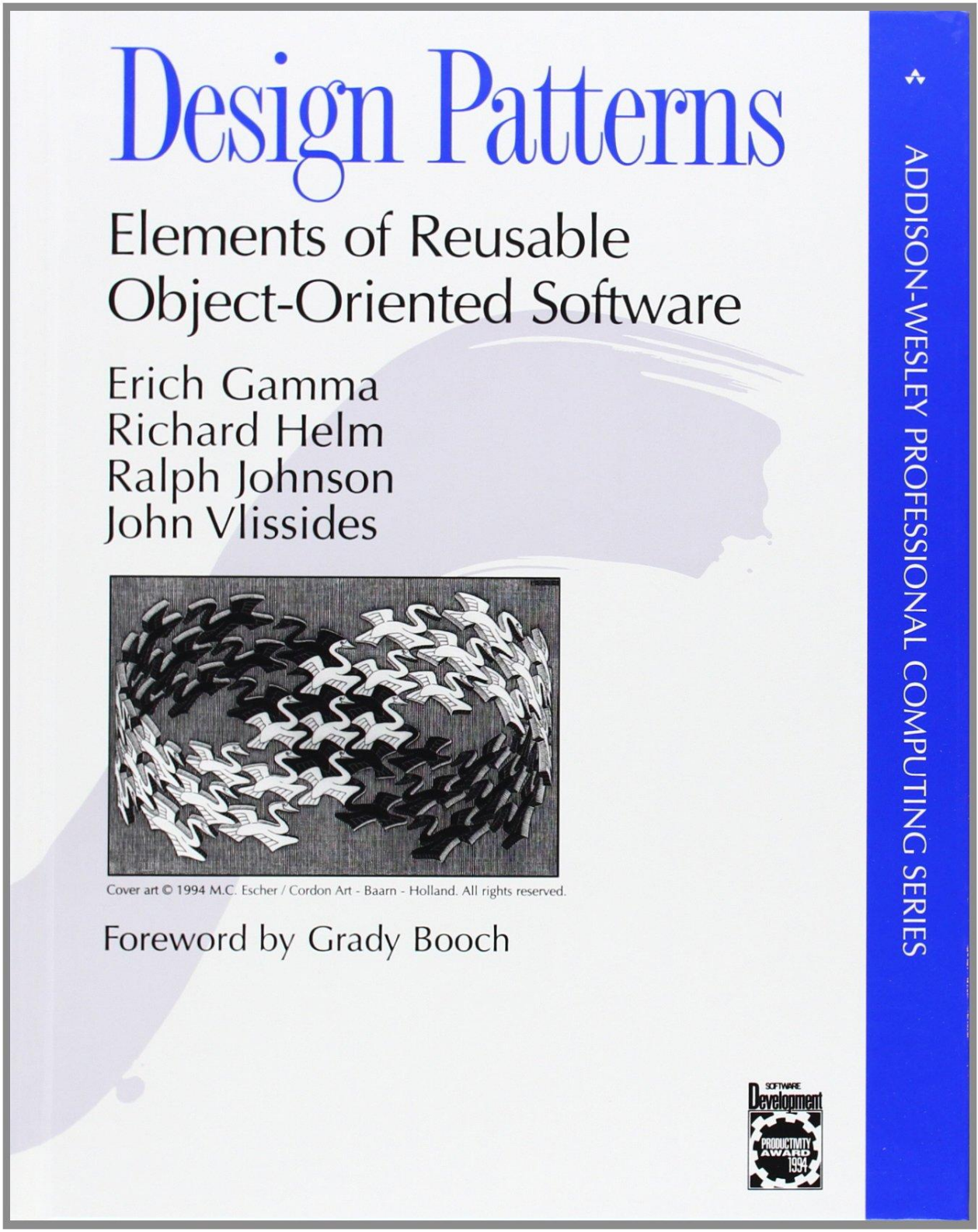
- Livros
  - *The Timeless Way of Building*
  - *A Pattern Language: Towns, Buildings, and Construction*
  - serviram de inspiração para os desenvolvedores de software.

Catálogo de soluções

- Um padrão encerra o conhecimento de uma pessoa muito experiente em um determinado assunto de uma forma que este conhecimento pode ser transmitido para outras pessoas menos experientes.
- Desde 1995, o desenvolvimento de software passou a ter o seu primeiro catálogo de soluções para projeto de software: o livro GoF.

Gang of Four (GoF)

- E. Gamma and R. Helm and R. Johnson and J. Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
-
-



Gang of Four (GoF)

- Passamos a ter um vocabulário comum para conversar sobre projetos de software.
- Soluções que não tinham nome passam a ter nome.
- Ao invés de discutirmos um sistema em termos de pilhas, filas, árvores e listas ligadas, passamos a falar de coisas de muito mais alto nível como Fábricas, Fachadas, Observador, Estratégia, etc.
- A maioria dos autores eram entusiastas de Smalltalk, principalmente o Ralph Johnson.
- Mas acabaram baseando o livro em C++ para que o impacto junto à comunidade de fosse maior. E o impacto foi enorme, o livro vendeu centenas de milhares de cópias.

O Formato de um padrão

- Todo padrão inclui
  - Nome
  - Problema
  - Solução
  - Conseqüências / Forças
  -
  -
- Na aula de hoje vamos nos concentrar nos padrões GoF.

O Formato dos padrões no GoF

- Nome (inclui número da página)
  - um bom nome é essencial para que o padrão caia na boca do povo
- Objetivo / Intenção
- Também Conhecido Como
- Motivação
  - um cenário mostrando o problema e a necessidade da solução
- Aplicabilidade
  - como reconhecer as situações nas quais o padrão é aplicável
- Estrutura
  - uma representação gráfica da estrutura de classes do padrão (usando OMT91) em, às vezes, diagramas de interação (Booch 94)
- Participantes
  - as classes e objetos que participam e quais são suas responsabilidades
- Colaborações
  - como os participantes colaboram para exercer as suas responsabilidades

O Formato dos padrões no GoF

- Conseqüências
  - vantagens e desvantagens, *trade-offs*
- Implementação
  - com quais detalhes devemos nos preocupar quando implementamos o padrão
  - aspectos específicos de cada linguagem
- Exemplo de Código
  - no caso do GoF, em C++ (a maioria) ou Smalltalk
- Usos Conhecidos
  - exemplos de sistemas reais de domínios diferentes onde o padrão é utilizado
- Padrões Relacionados
  - quais outros padrões devem ser usados em conjunto com esse
  - quais padrões são similares a este, quais são as dierenças

Tipos de Padrões de Projeto

- Categorias de Padrões do GoF
  - Padrões de Criação
  - Padrões Estruturais
  - Padrões Comportamentais

- Vamos ver um exemplo de cada um deles.

- Na aula de hoje:
  - Simple Fsctory, Factory Method, Fábrica Abstrata (Abstract Factory) - padrão de Criação de objetos

Criação de Objetos

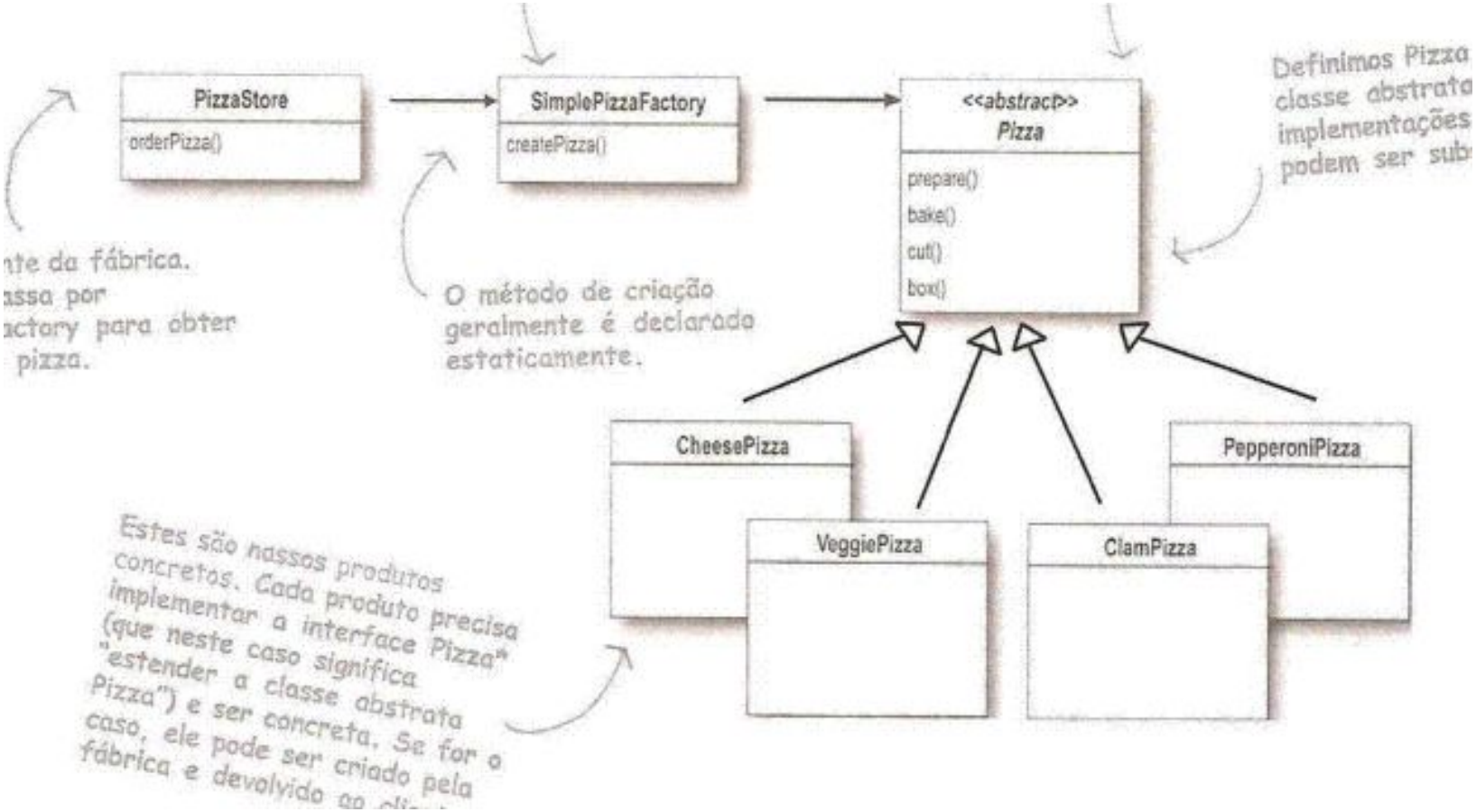
- Como lidar com a criação de objetos no código?
- Qual a forma mais simples e imediata de fazermos isso? Vamos tentar escrever uma **classe Pizzaria que cria pizzas diferentes baseado em parâmetros** passados a ela pelo Cliente.
- Pizza pode ser uma classe abstrata, com subclasses explicitando cada sabor de pizza. Lembre-se que em geral não é boa prática derivar de classes concretas!

Simple Factory

- Pode não ser boa ideia colocarmos código dedicado a instanciamento de objetos junto com outro código. Deve haver alguma solução melhor organizada que isso!
- Podemos pensar em criar uma classe apenas para criar instâncias! Chamemos isso de **SimpleFactory**
- Uma das vantagens é que a SimpleFactory pode ter seus métodos chamados por vários clientes agora. Também deixamos a criação, que pode ser complexa, delegada a um objeto específico
- Muitas vezes a Simple Factory é uma classe estática (`static`)
- Simple Factory é útil, mas muita gente não considera ela um padrão :(

Simple Factory

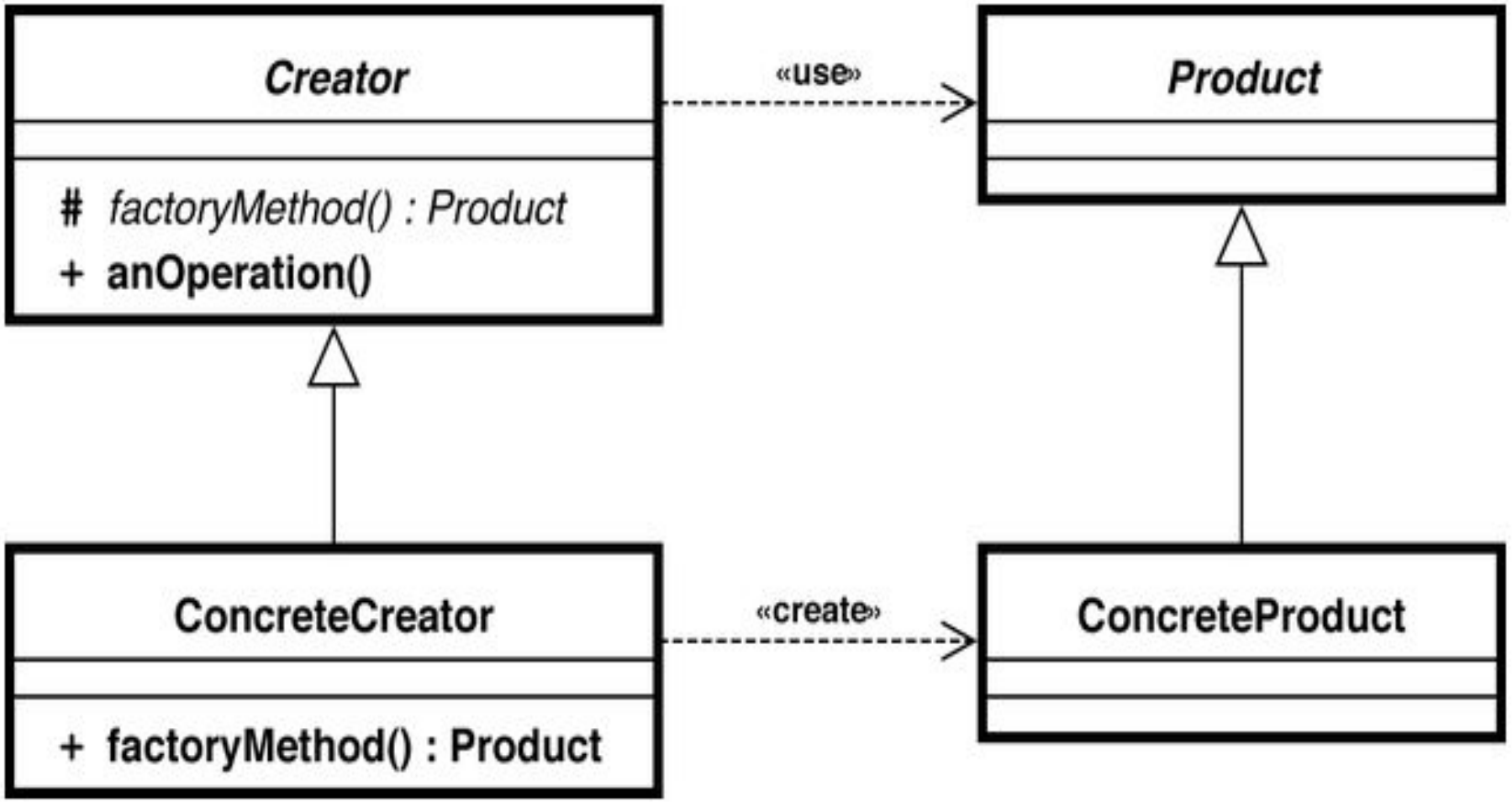
- Como fica o diagrama de classes, então?



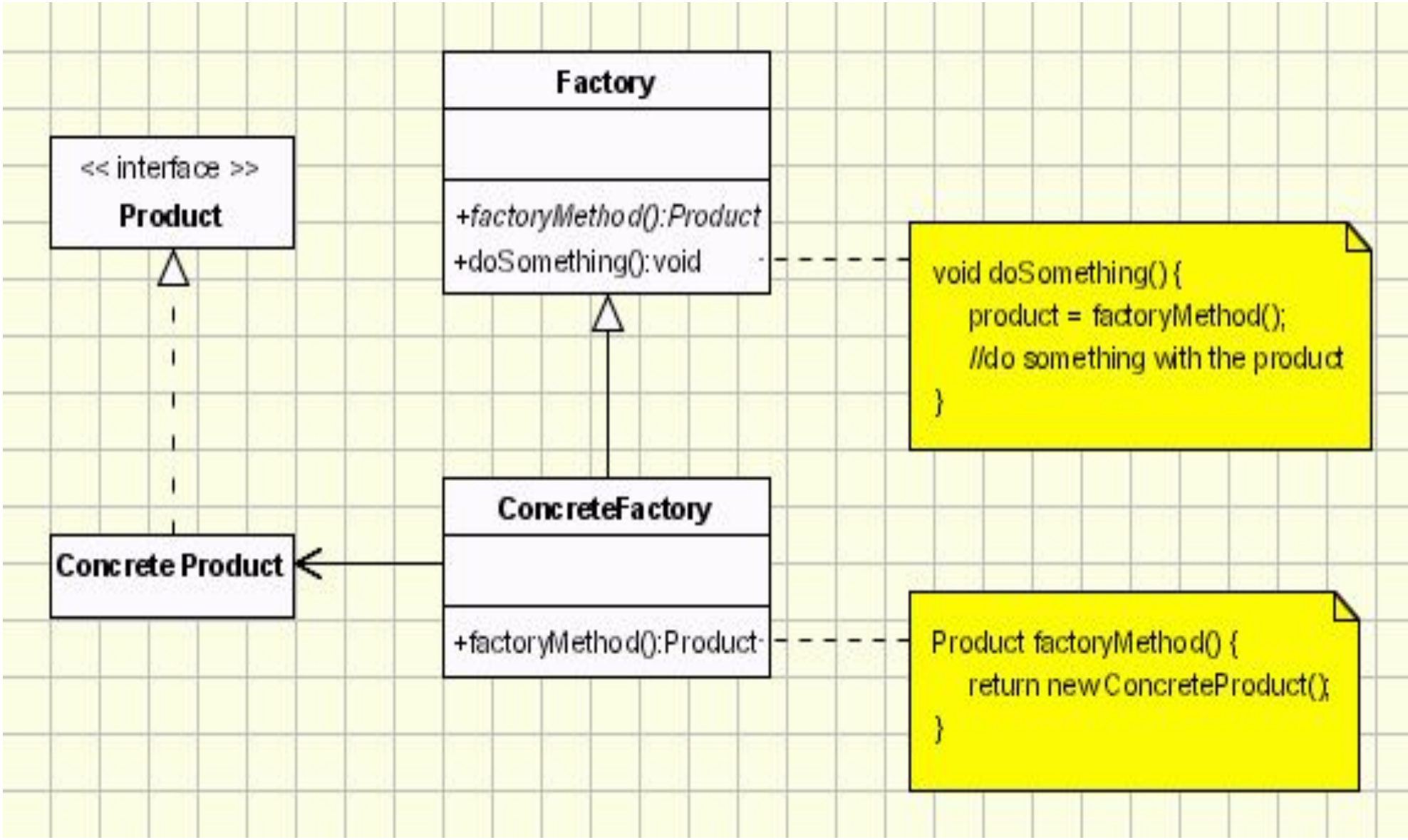
Factory Method

- O factory Method é um padrão creacional, ou seja, seu propósito é a criação de novos objetos
- Ele define uma interface para criar um objeto, **mas deixa que as subclasses decidam que classe será instanciada**.
- Note que o Factory Method chamado deve ser o da subclasse correta. Se quero carregar um GIF, preciso de uma instância de GIFFactory (JPGFactory não serviria) e preciso chamar seu método loadImage()
- Problema: **`new`** não é polimórfico
-

Factory Method



Factory Method



Factory Method

- Veja como o método encapsula o conhecimento sobre o objeto dentro de cada criador!
- Falamos que o Factory Method deferencia a decisão do tipo de objeto para as subclasses
- Se o Factory Method recebe um parâmetro para que a Factory tome decisões, chamamos de Factory Method parametrizado

Factory Method

Factory Method

Factory Method

- Princípios do bom desenvolvimento OO:

  Programe para interfaces, não para implementações

  Dependa de abstrações, não dependa de Objetos concretos

Fábrica Abstrata
Abstract Factory

Objetivo:

- Prover uma interface para criação de famílias de objetos relacionados sem especificar sua classe concreta
- Melhor organizado que o Factory Method
-
-

Abstract Factory - Motivação

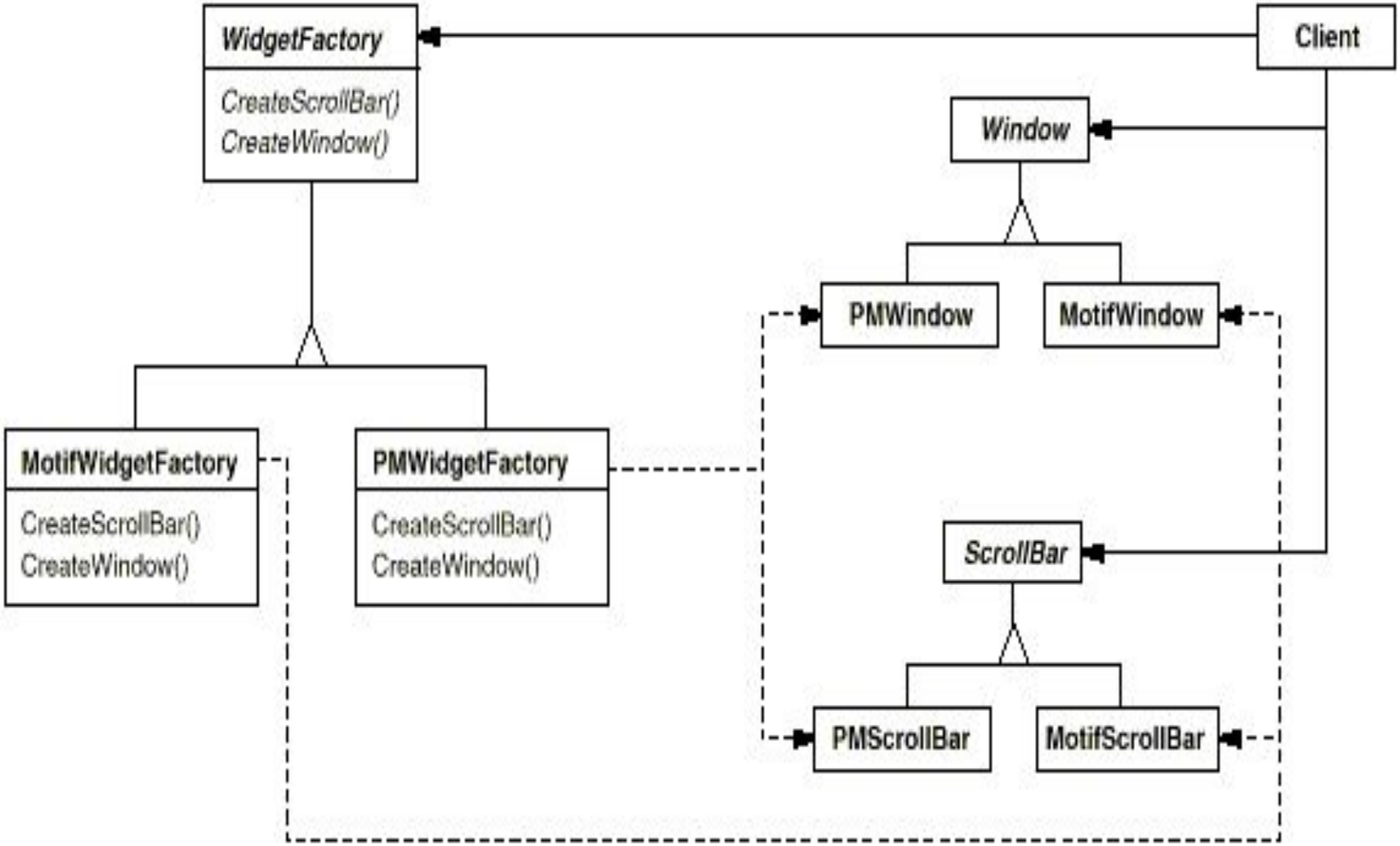
- Considere uma aplicação com interface gráfica que é implementada para plataformas diferentes (Motif para UNIX e outros ambientes como Qt, wxWidgets ou Swing para Windows e MacOS).
- As classes implementando os elementos gráficos não podem ser definidas estaticamente no código. Precisamos de uma implementação diferente para cada ambiente. Até em um mesmo ambiente, gostaríamos de dar a opção ao usuário de implementar diferentes aparências (*look-and-feels*).
- Podemos solucionar este problema definindo uma classe abstrata para cada elemento gráfico e utilizando diferentes implementações para cada aparência ou para cada ambiente.
- Ao invés de criarmos as classes concretas com o operador **`new`**, utilizamos uma Fábrica Abstrata para criar os objetos em tempo de execução.
- O código cliente não sabe qual classe concreta utilizamos.

Abstract Factory - Aplicabilidade

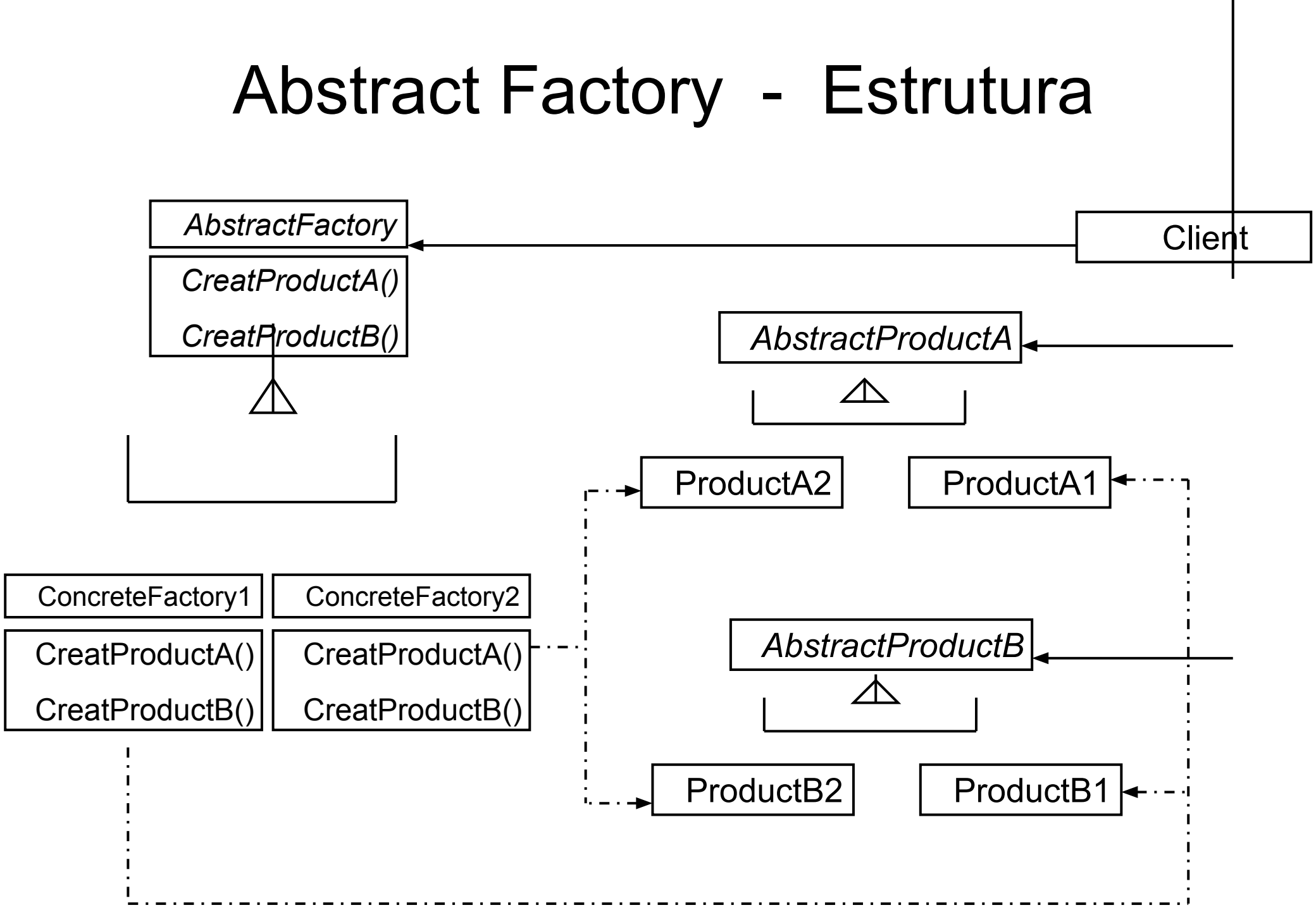
Use uma fábrica abstrata quando:

- um sistema deve ser independente da forma como seus produtos são criados e representados;
- um sistema deve poder lidar com uma família de vários produtos diferentes;
- você quer prover uma biblioteca de classes de produtos mas não quer revelar as suas implementações, quer revelar apenas suas interfaces.
- Padrão de Projeto também chamado de Kit

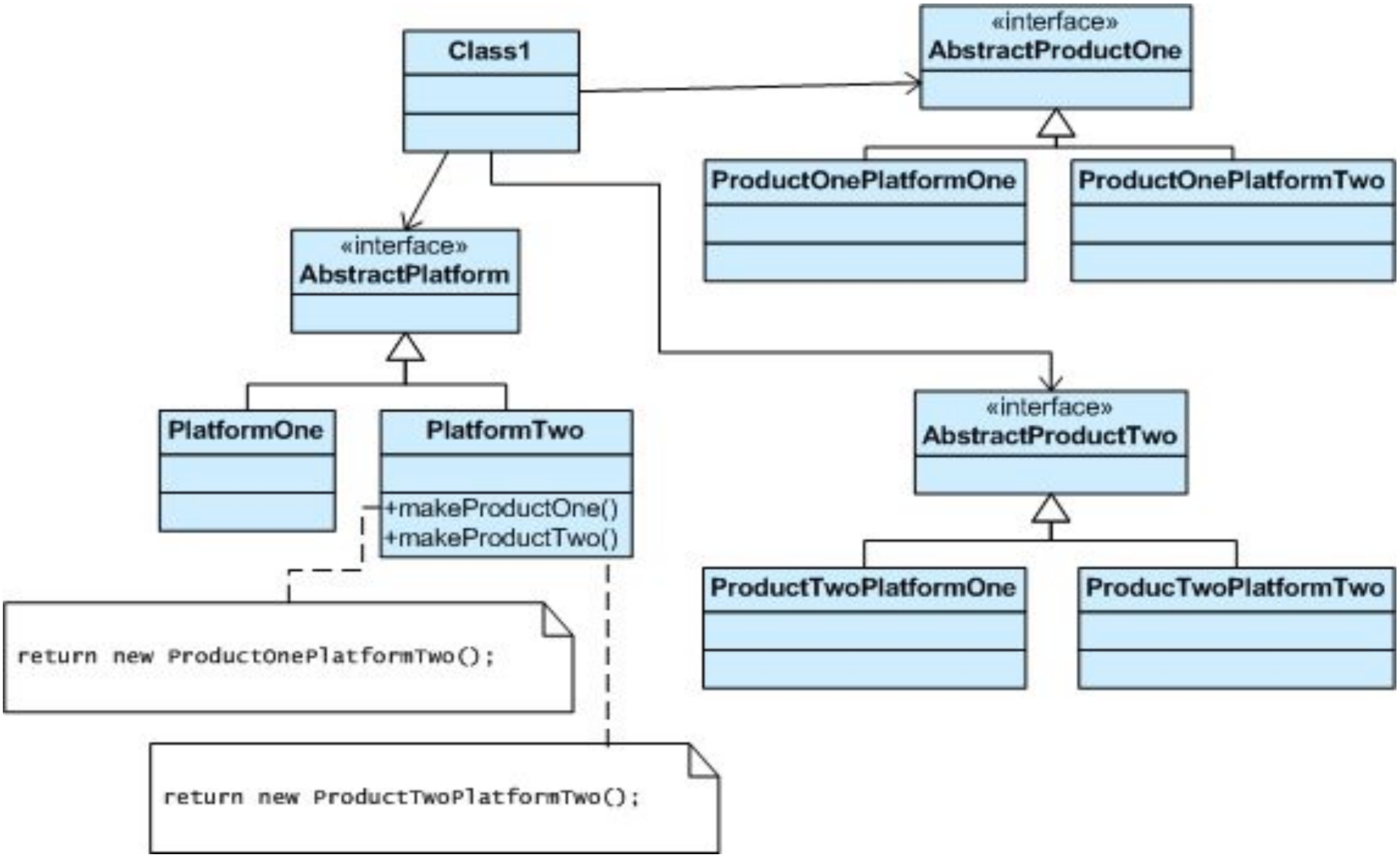
Abstract Factory - Estrutura



Abstract Factory - Estrutura



Abstract Factory - Estrutura



Abstract Factory - Participantes

- **`AbstractFactory`** (WidgetFactory)
- **`ConcreteFactory`** (MotifWidgetFactory, WindowsWidgetFactory)
- **`AbstractProduct`** (Window, ScrollBar)
- **`ConcreteProduct`** (MotifWindow, MotifScrollBar, WindowsWindow, WindowsScrollBar)
- **`Client`** - usa apenas as interfaces declaradas pela AbstractFactory e pelas classes AbstratProduct
-

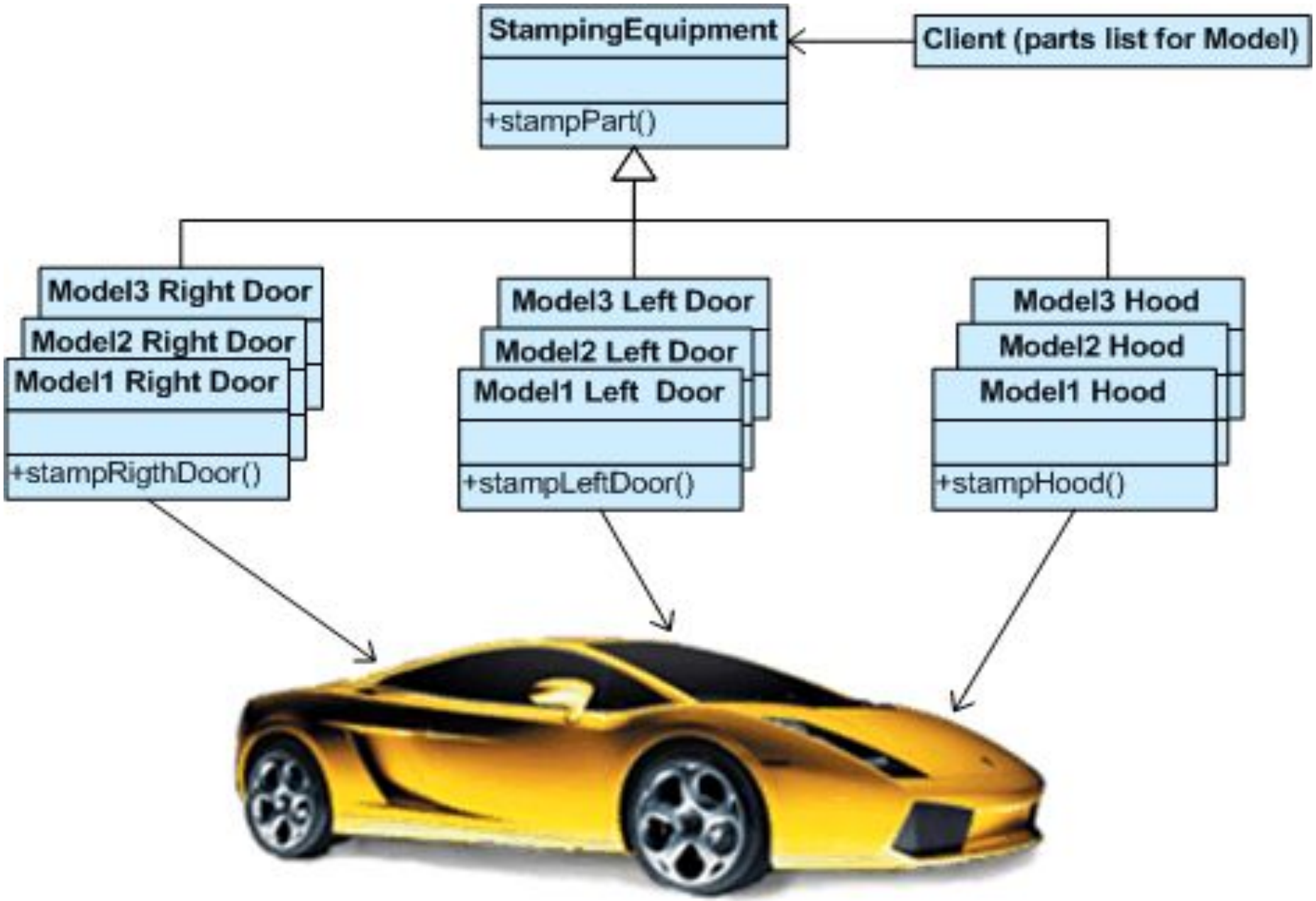
Abstract Factory - Colaborações

- Normalmente, apenas uma instância de **`ConcreteFactory`** é criada em tempo de execução.
- Esta instância cria objetos através das classes **`ConcreteProduct`** correspondentes a uma família de produtos.
- Uma **`AbstractFactory`** deixa a criação de objetos para as suas subclasses **`ConcreteFactory`**.

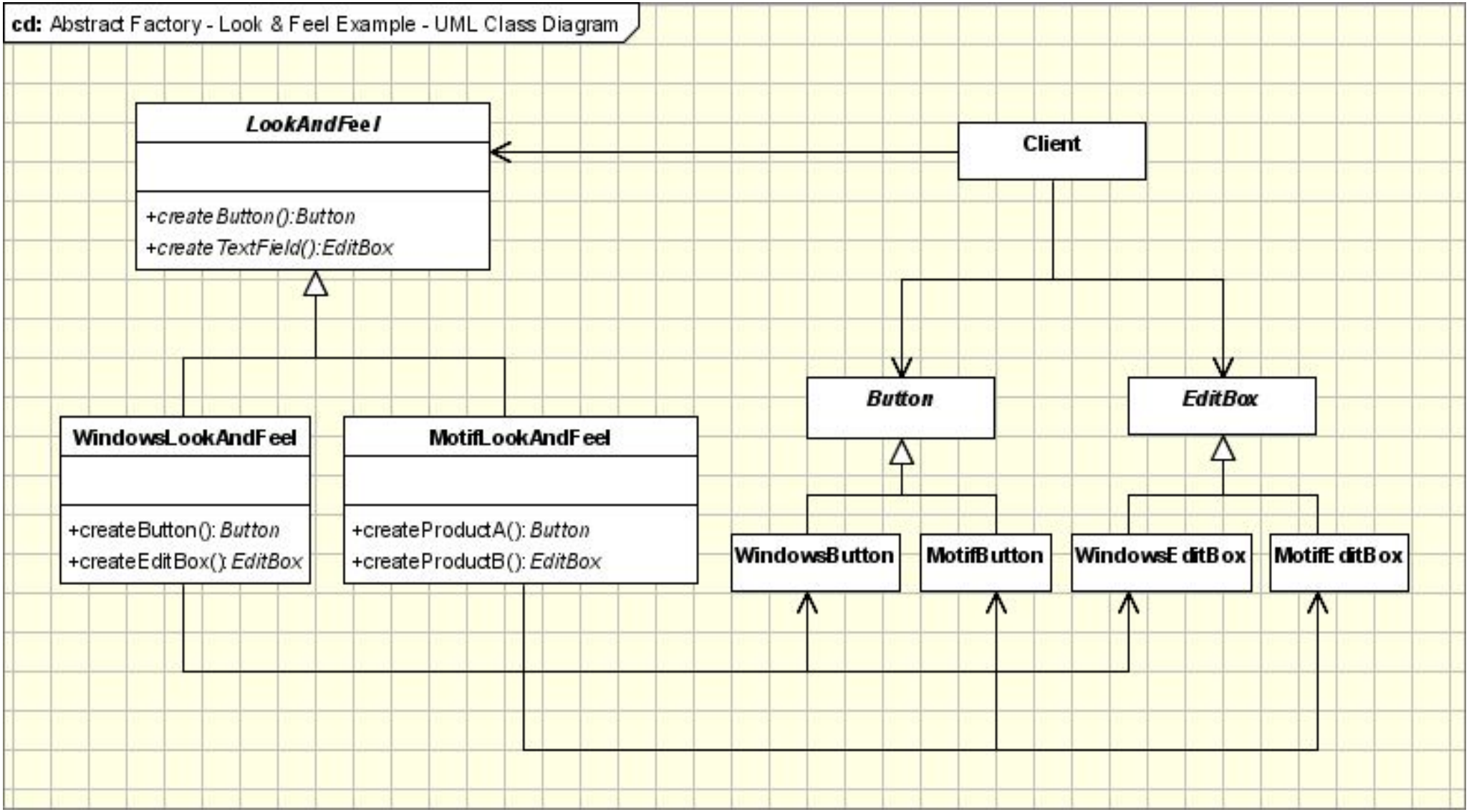
Abstract Factory - Características

- Às vezes (como no caso da interface gráfica) você só precisa instanciar um classe concreta em particular durante a execução
- A classe de uma Fábrica Concreta só aparece uma vez em uma aplicação: quando ela é instanciada
- Promove consistência de produtos. Uma família de produtos é criada junta e atrelada

Abstract Factory - Exemplo



Abstract Factory - Exemplo



Abstract Factory - Consequências

O padrão

1. **isola as classes concretas dos clientes**;
2. **facilita a troca de famílias de produtos** (basta trocar uma linha do código pois a criação da fábrica concreta aparece em um único ponto do programa);
3. **promove a consistência de produtos** (não há o perigo de misturar objetos de famílias diferentes);
4. **dificulta a criação de novos produtos ligeiramente diferentes** (pois temos que modificar a fábrica abstrata e todas as fábricas concretas).

Abstract Factory - Implementação

Na fábrica abstrata, cria-se um método fábrica para cada tipo de produto. Cada fábrica concreta implementa o código que cria os objetos de fato.

Se tivermos muitas famílias de produtos, teríamos um excesso de classes “fábricas concretas”.

Para resolver este problema, podemos usar o **`Prototype`** (outro padrão): criamos um dicionário mapeando tipos de produtos em instâncias prototípicas destes produtos. Veremos isso mais adiante!

Os 23 Padrões do GoF
Criação

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

Os 23 Padrões do GoF
Estruturais

- Adapter
- Bridge
- Composite
- Decorator
- Façade
- Flyweight
- Proxy

Os 23 Padrões do GoF Comportamentais

- Chain of Responsibility
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method
- Visitor

Agradecimentos

Prof. Fabio Kon – IME/USP por grande parcela do material