Universidade de São Paulo
Brasil

Aula 08 – Padrões de Projeto de Software

Reutilização de “ideias” boas no desenvolvimento de software

MAC0321 - Laboratório de Programação Orientada a Objetos

Professor: Marcelo Finger (mfinger@ime.usp.br)

Departamento de Ciência da Computação
Instituto de Matemática e Estatística

Tópicos

1. Padrões
2. Troca de Informações
3. Formato de um padrão
4. Exemplo: Fábrica Abstrata
5. Os 23 padrões do GoF

Padrões de Projeto de Software Orientado a Objetos

Padrões de Projeto de Software OO

Também conhecidos como

- *Padrões de Desenho de Software OO*
- ou simplesmente como *Padrões*.

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 implementá-la duas vezes da mesma forma."*

Livros serviram de inspiração para os desenvolvedores de software. Ex:

- *The Timeless Way of Building*
- *A Pattern Language: Towns, Buildings, and Construction*

Catálogo de soluções

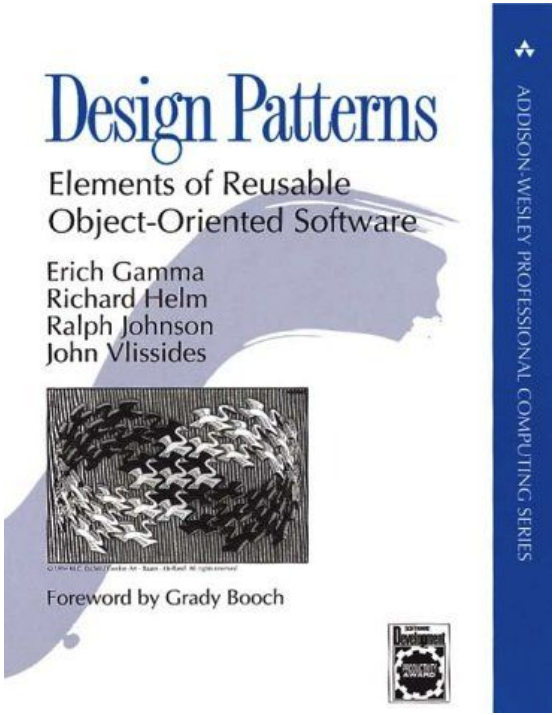
- Um padrão encerra o conhecimento de uma pessoa muito experiente em um determinado assunto de uma forma que este conhecimento pode ser transmitido para outras pessoas menos experientes.
- Outras ciências (p.ex. química) e engenharias possuem catálogos de soluções.
- Desde 1995, o desenvolvimento de 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.



GoF e a Troca de Informações

- 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 OO fosse maior. E o impacto foi enorme, o livro vendeu centenas de milhares de cópias.

Formato de um Padrão

O Formato de um padrão

- Todo padrão inclui
  - Nome
  - Problema
  - Solução
  - Consequências / Forças
- Existem outros tipos de padrões mas na aula de hoje vamos nos concentrar nos do 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 (cont)

- Consequências
  - 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 diferenças

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:
  - Fábrica Abstrata (Abstract Factory (87))
    - padrão de Criação de objetos

Exemplo de Padrão

Fábrica Abstrata
Abstract Factory (87)

Objetivo:

- prover uma interface para criação de famílias de objetos relacionados sem especificar sua classe concreta.

Abstract Factory: Motivação

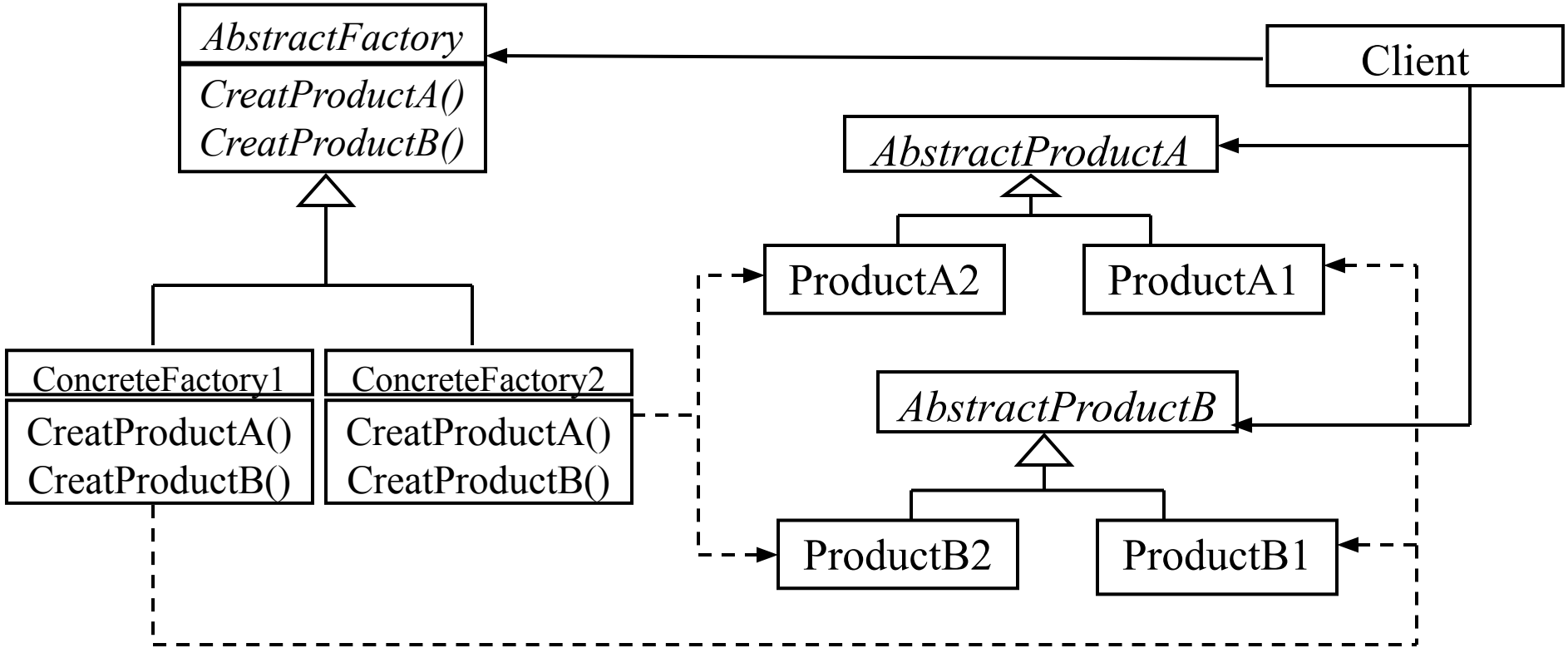
- Considere uma aplicação com interface gráfica que é implementada para plataformas diferentes (GTK para UNIX/Gnome e outros ambientes para Windows, MacOS, iOS e Android).
- As classes implementando os elementos gráficos não podem ser definidas estaticamente no código. Precisamos de uma implementação diferente para cada ambiente. Até em um mesmo ambiente, gostaríamos de dar a opção ao usuário de implementar diferentes aparências (*look-and-feel*).
- Podemos solucionar este problema definindo uma classe abstrata para cada elemento gráfico e utilizando diferentes implementações para cada aparência ou para cada ambiente.
- Em vez 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.

Abstract Factory: Estrutura



Abstract Factory: Participantes

- **`AbstractFactory`** (WidgetFactory)
- **`ConcreteFactory`** (GTKWidgetFactory, WindowsWidgetFactory)
- **`AbstractProduct`** (Window, ScrollBar)
- **`ConcreteProduct`** (GTKWindow, GTKScrollBar, 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: 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

1. Fábricas abstratas em geral são implementadas como Singleton(127).
2. 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.
3. Se tivermos muitas famílias de produtos, teríamos um excesso de classes “fábricas concretas”.

   Para resolver este problema, podemos usar o padrão **`Prototype`** (117): criamos um dicionário mapeando tipos de produtos em instâncias prototípicas destes produtos.

   Então, sempre que precisarmos criar um novo produto pedimos à sua instância prototípica que crie um clone (usando um método como clone() ou copy()).

Abstract Factory: Implementação

4. Definindo fábricas extensíveis.
   - normalmente, cada tipo de produto tem o seu próprio método fábrica; isso torna a inclusão de novos produtos difícil.
   - solução: usar apenas um método fábrica
     - `Product make (string thingToBeMade)`
   - isso aumenta a flexibilidade mas torna o código menos seguro (não teremos verificação de tipos pelo compilador).

- *Abstract Factory -*

Exemplos de Código

- O GoF contém exemplos em C++ e Smalltalk
- Ver exemplo fornecido para Java

Abstract Factory: Usos Conhecidos

- **InterViews** usa fábricas abstratas para encapsular diferentes tipos de aparências para sua interface gráfica
- **ET++** usa fábricas abstratas para permitir a fácil portabilidade para diferentes ambientes de janelas (XWindows e SunView, por exemplo)
- Sistema de captura e reprodução de vídeo feito na UIUC usa fábricas abstratas para permitir portabilidade entre diferentes placas de captura de vídeo.
- Em linguagens dinâmicas como Smalltalk (e talvez em POO em geral) classes podem ser vistas como fábricas de objetos.

Abstract Factory

Padrões Relacionados

- Fábricas abstratas são normalmente implementadas com métodos fábrica (**`FactoryMethod`** (107)) mas podem também ser implementados usando **`Prototype`** (117).
- O uso de protótipos é particularmente importante em linguagens não dinâmicas como C++ e em linguagens "semi-dinâmicas" como Java.
- Uma fábrica concreta é normalmente um **`Singleton`** (127)

Os 23 Padrões do GoF

Criação

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

Estruturais

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

Comportamentais

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

Para o aconchego do lar

- Dar uma olhada no GoF
  - a biblioteca possui algumas cópias
- Buscar por “GoF patterns” no google
- Tb buscar por outros tipos de padrões (pós-GoF)
  - arquiteturais
  - de análise
  - anti-padrões
  - etc.

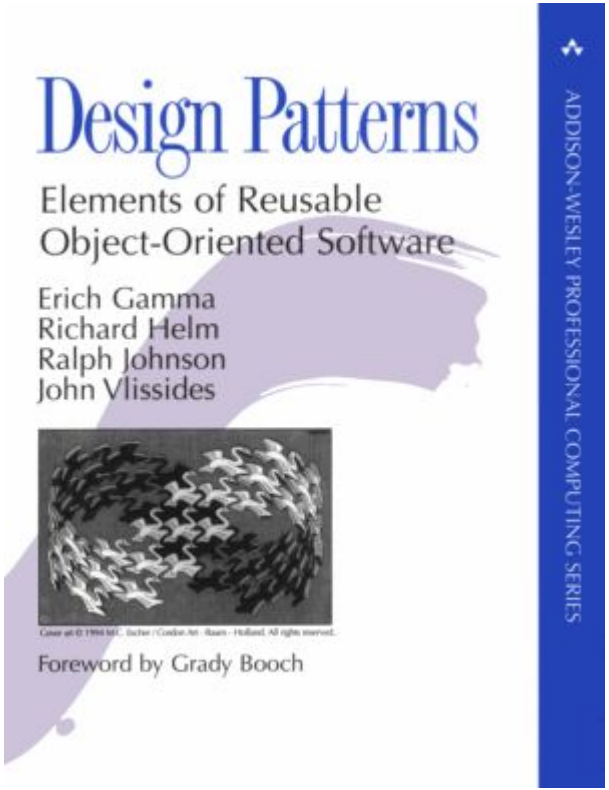
Recapitulando

- Voltando ao Christopher Alexander:

  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 implementá-la duas vezes da mesma forma.

- Talvez a última parte não seja sempre desejável.

Bibliografia



Lista de exercícios

No computador com o Eclipse

Entrega até o final do dia

MAC321
Lab POO

- Professor: Marcelo Finger

  E-mail: mfinger@ime.usp.br