Modelagem da Arquitetura

Padrões e Frameworks

Padrões

- Os sistemas “bem estruturados” utilizam padrões

- Projetistas experientes (re)utilizam soluções bem sucedidas no passado

- Durante as duas últimas décadas, surgiu uma “comunidade” voltada a padrões

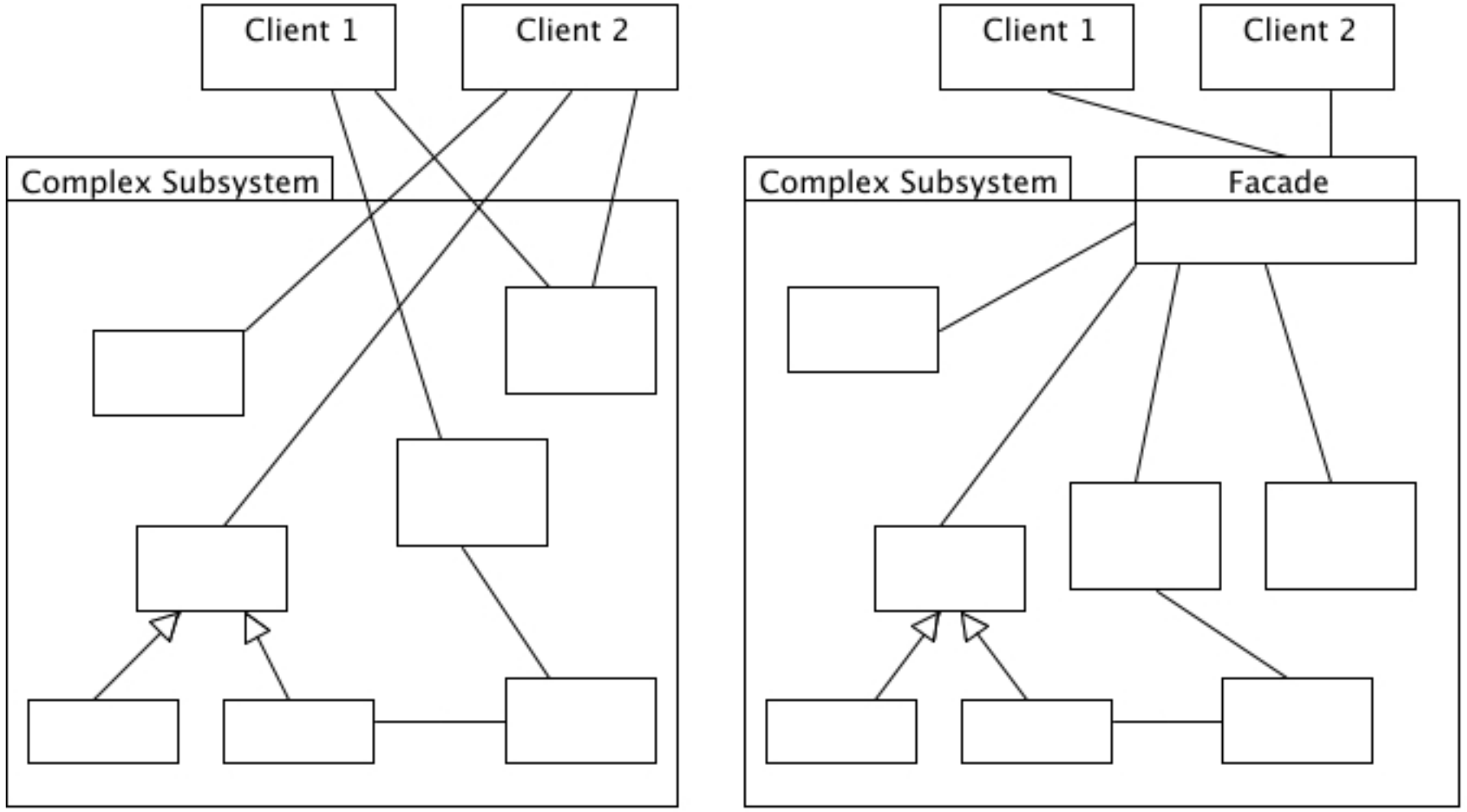
Padrões

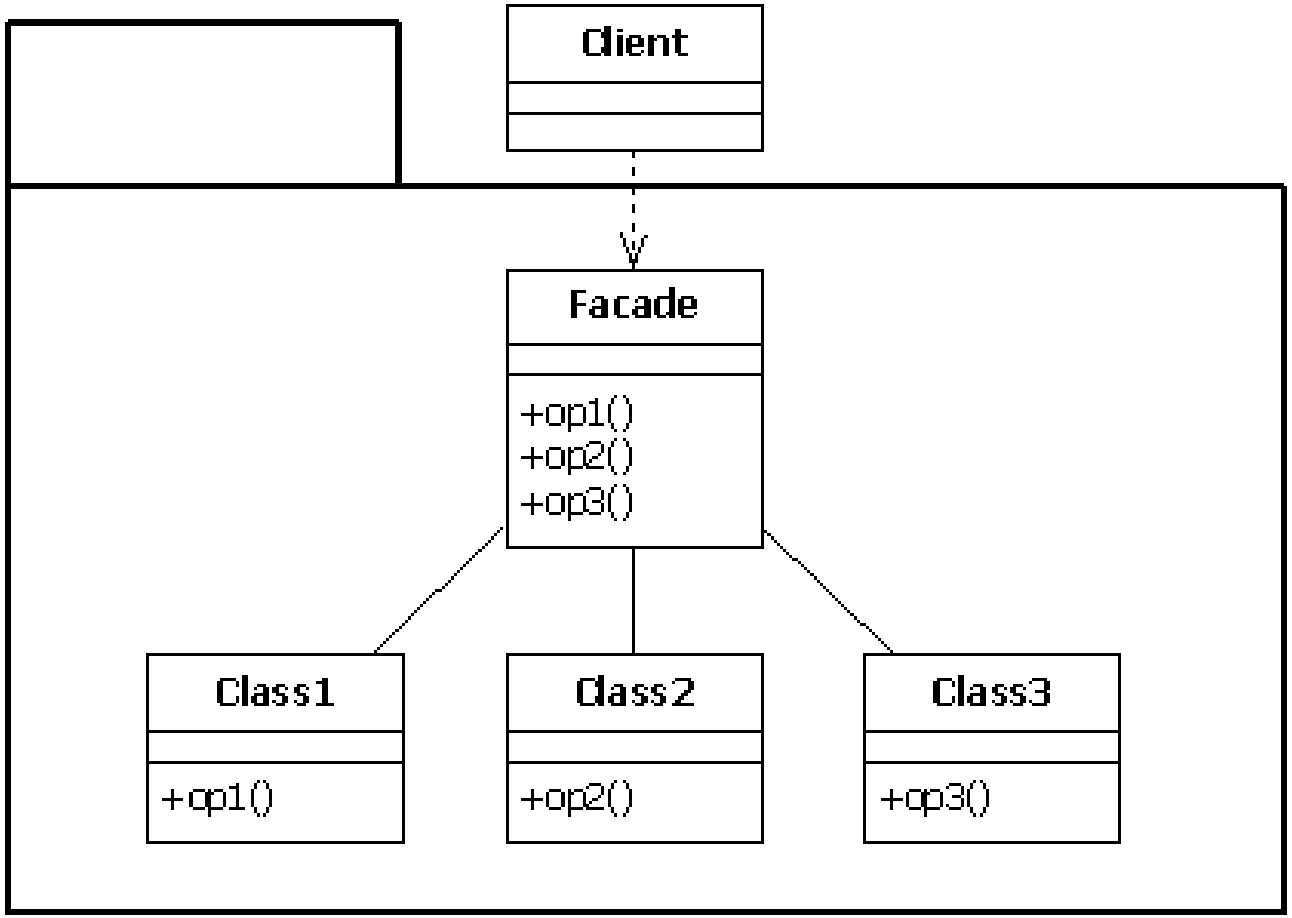
- Um padrão descreve uma solução comum para um problema recorrente (em um dado contexto)
  - Um mecanismo é um padrão de projeto aplicado a um conjunto de classes
  - Um framework é tipicamente um padrão de arquitetura que fornece *template* extensível para aplicações em um domínio

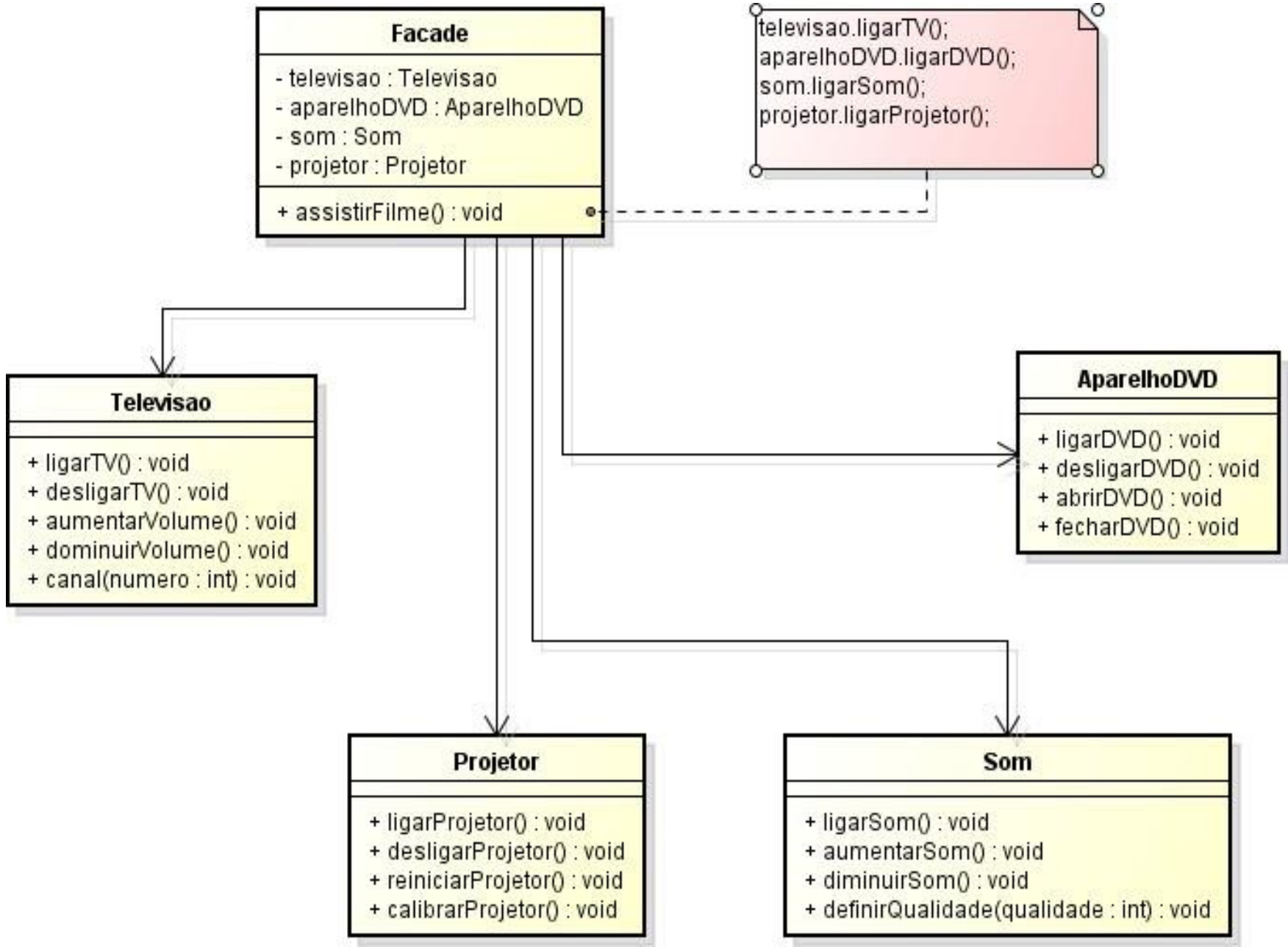
Padrões de Projeto

- Padrões sistematizam soluções, incluindo:
  - Nome
  - Problema
  - Solução
  - Consequência
  - Exemplo, ...

- A representação de um padrão de projeto pode ser:
  - Abstrata – para facilitar o entendimento
  - Diagramas de classes e sequência UML – funcionamento

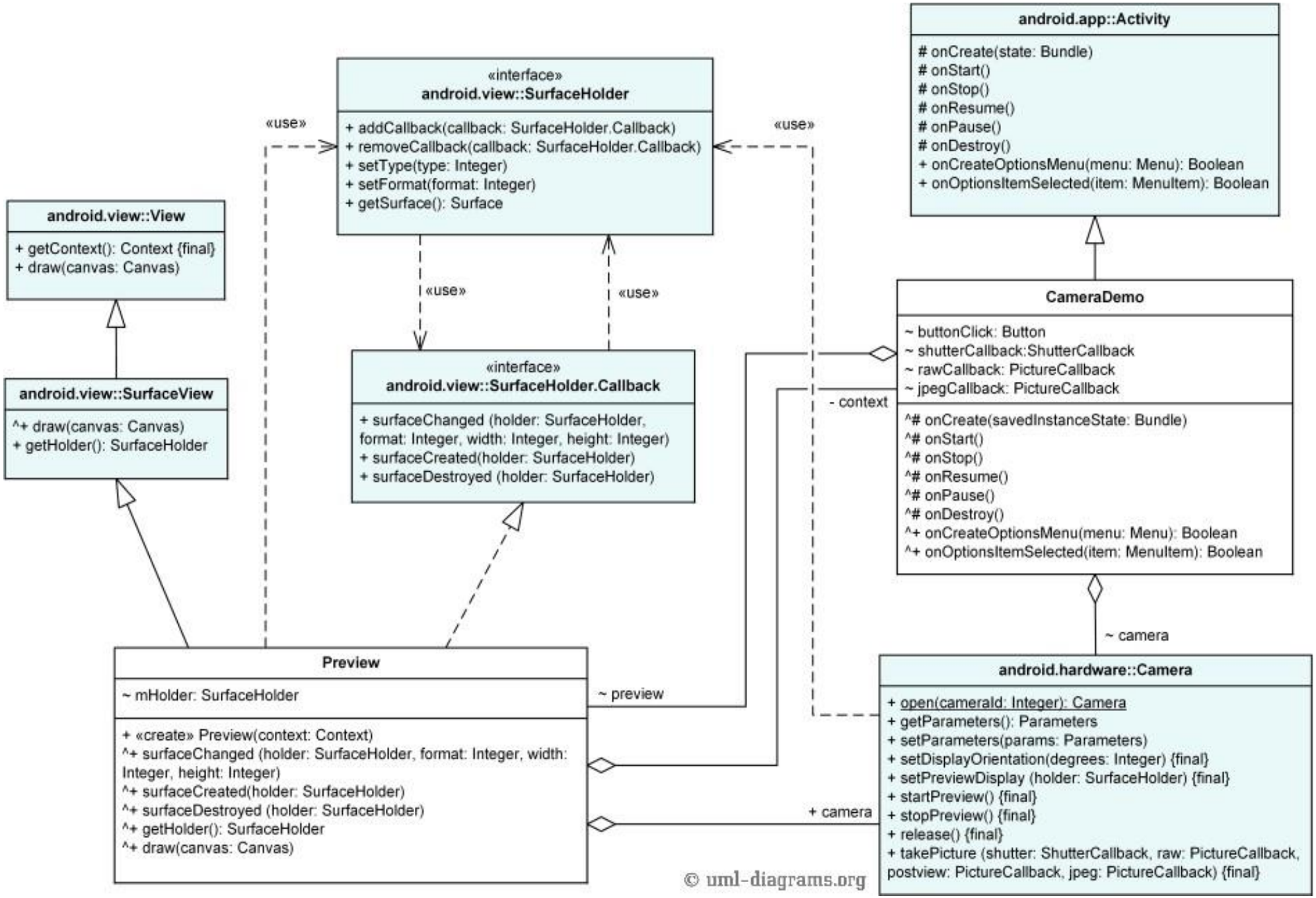

Client 1
Client 2
Complex Subsystem
Client 1
Client 2
Complex Subsystem
Facade


Client
Facade
+op1()
+op2()
+op3()
Class1
+op1()
Class2
+op2()
Class3
+op3()


Facade
- televisao : Televisao
- aparelhoDVD : AparelhoDVD
- som : Som
- projetor : Projetor
+ assistirFilme() : void
televisao.ligarTV();
aparelhoDVD.ligarDVD();
som.ligarSom();
projetor.ligarProjetor();
Televisao
+ ligarTV() : void
+ desligarTV() : void
+ aumentarVolume() : void
+ dominuirVolume() : void
+ canal(numero : int) : void
AparelhoDVD
+ ligarDVD() : void
+ desligarDVD() : void
+ abrirDVD() : void
+ fecharDVD() : void
Projetor
+ ligarProjetor() : void
+ desligarProjetor() : void
+ reiniciarProjetor() : void
+ calibrarProjetor() : void
Som
+ ligarSom() : void
+ aumentarSom() : void
+ diminuirSom() : void
+ definirQualidade(qualidade : int) : void

Frameworks

- Usualmente baseado em padrões, mas já voltados para uma linguagem de programação


android.app::Activity
onCreate(state: Bundle)
onStart()
onStop()
onResume()
onPause()
onDestroy()
+ onCreateOptionsMenu(menu: Menu): Boolean
+ onOptionsItemSelected(item: MenuItem): Boolean
«interface»
android.view::SurfaceHolder
+ addCallback(callback: SurfaceHolder.Callback)
+ removeCallback(callback: SurfaceHolder.Callback)
+ setType(type: Integer)
+ setFormat(format: Integer)
+ getSurface(): Surface
«use»
«use»
«use»
«use»
android.view::View
+ getContext(): Context {final}
+ draw(canvas: Canvas)
CameraDemo
~ buttonClick: Button
~ shutterCallback:ShutterCallback
~ rawCallback: PictureCallback
~ jpegCallback: PictureCallback
^# onCreate(savedInstanceState: Bundle)
^# onStart()
^# onStop()
^# onResume()
^# onPause()
^# onDestroy()
^+ onCreateOptionsMenu(menu: Menu): Boolean
^+ onOptionsItemSelected(item: MenuItem): Boolean
- context
«interface»
android.view::SurfaceHolder.Callback
+ surfaceChanged (holder: SurfaceHolder, format: Integer, width: Integer, height: Integer)
+ surfaceCreated(holder: SurfaceHolder)
+ surfaceDestroyed (holder: SurfaceHolder)
android.view::SurfaceView
^+ draw(canvas: Canvas)
+ getHolder(): SurfaceHolder
~ camera
Preview
~ mHolder: SurfaceHolder
~ preview
+ «create» Preview(context: Context)
^+ surfaceChanged (holder: SurfaceHolder, format: Integer, width: Integer, height: Integer)
^+ surfaceCreated(holder: SurfaceHolder)
^+ surfaceDestroyed (holder: SurfaceHolder)
^+ getHolder(): SurfaceHolder
^+ draw(canvas: Canvas)
android.hardware::Camera
+ open(cameraId: Integer): Camera
+ getParameters(): Parameters
+ setParameters(params: Parameters)
+ setDisplayOrientation(degrees: Integer) {final}
+ setPreviewDisplay (holder: SurfaceHolder) {final}
+ startPreview() {final}
+ stopPreview() {final}
+ release() {final}
+ takePicture (shutter: ShutterCallback, raw: PictureCallback, postview: PictureCallback, jpeg: PictureCallback) {final}
+ camera
© uml-diagrams.org

Arquitetura de Software

Padrões de Projetos

Nairon Neri Silva

Sumário

- Introdução
- Catálogo de Soluções
- O Formato de um Padrão
- Tipos de Padrões de Projeto
    - Abstract Factory

Introdução

- Também conhecidos como
  - Padrões de Projeto de Software OO
  - ou simplesmente como Padrões

- 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

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

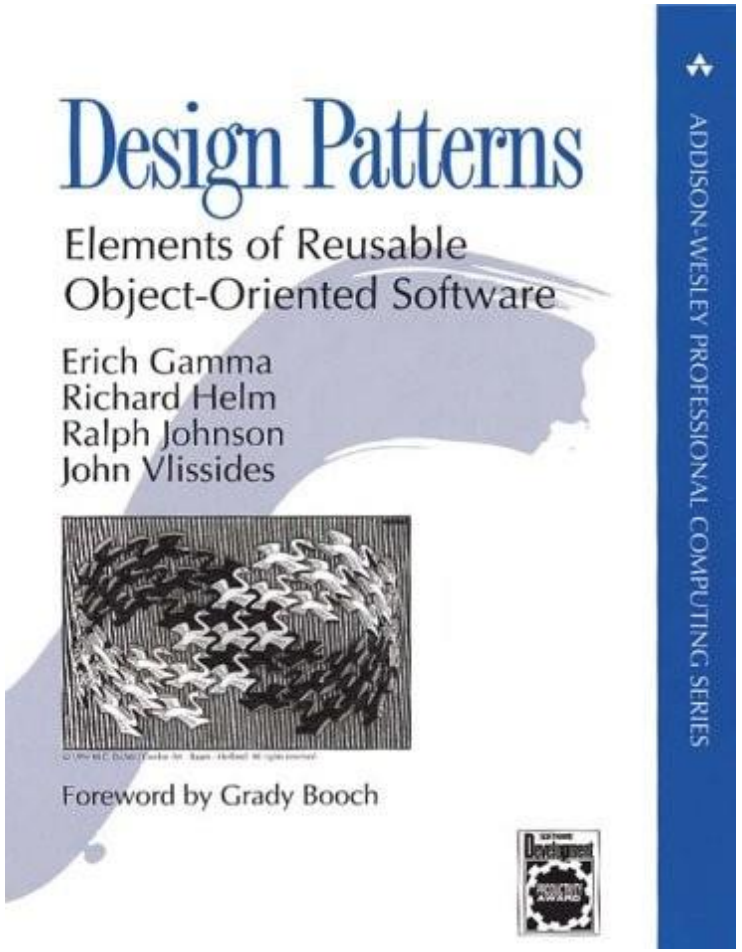
  Serviram de inspiração para os desenvolvedores de software

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
- 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.



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.

Gang of Four (GoF)

- 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 CC 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
  - Consequências / Forças

- Existem outros tipos de padrões mas na aula de hoje vamos nos concentrar no 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

O Formato dos padrões no GoF

- **Colaborações**
  - como os participantes colaboram para exercer as suas responsabilidades
- **Consequências**
  - vantagens e desvantagens, trade-offs (porque escolher uma ou outra)
- **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
  1. Padrões Criacionais
  2. Padrões Estruturais
  3. Padrões Comportamentais

- Na aula de hoje:
  - Padrão de Criação de objetos: Fábrica Abstrata (*Abstract Factory*)

Padrões de Projeto - GoF

Abstract Factory (Fábrica Abstrata)

Fábrica Abstrata

- Objetivo:

  Prover uma interface para criação de famílias de objetos relacionados sem especificar sua classe concreta

Fábrica Abstrata – Motivação

- Considere uma aplicação com interface gráfica que é implementada para plataformas diferentes (Motif para UNIX e outros ambientes 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-feel*)

Fábrica Abstrata – Motivação

- Podemos solucionar o referido problema definindo uma classe abstrata para cada elemento gráfico e utilizando diferentes implementações para cada aparência ou para cada ambiente

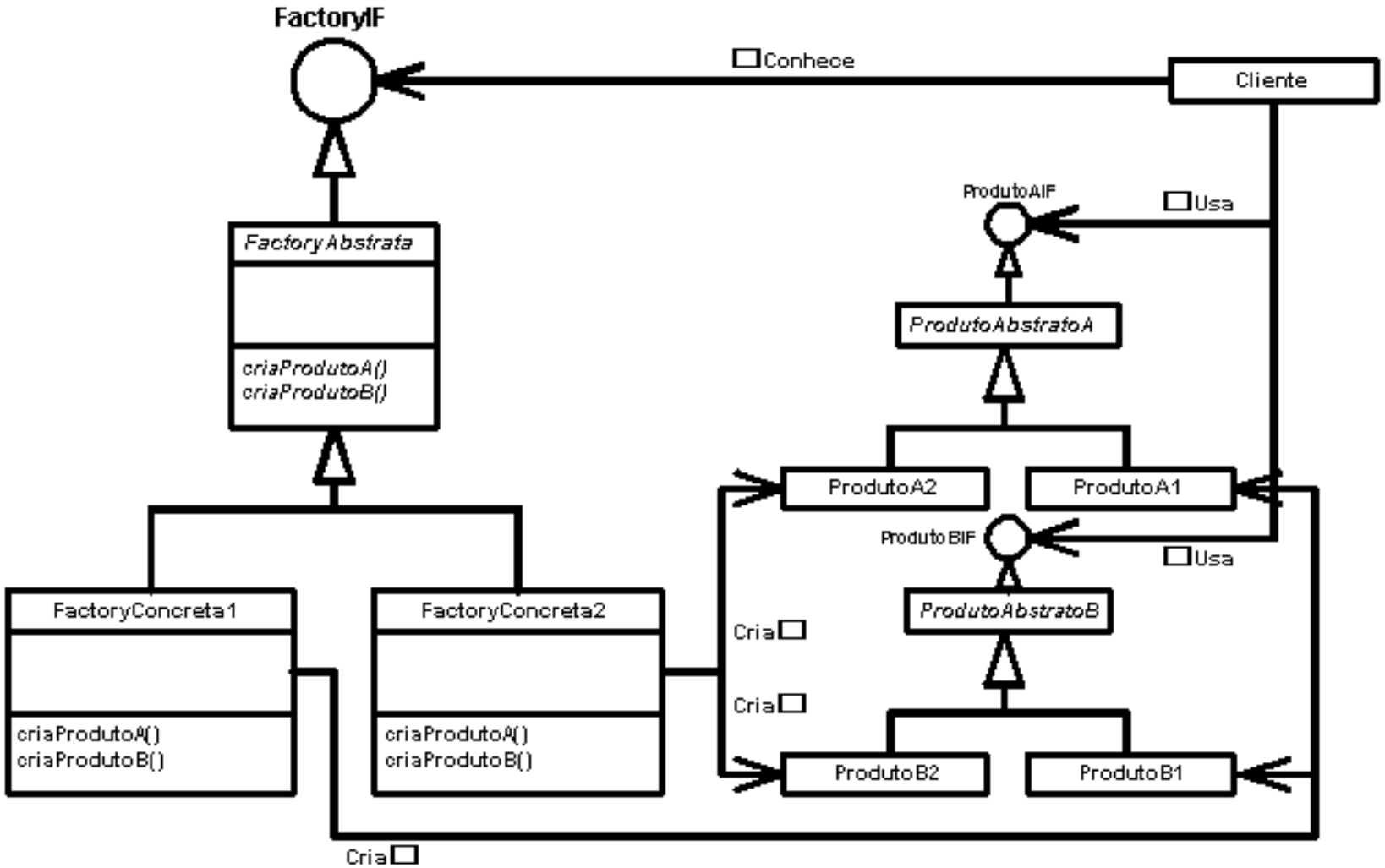
- Ao invés de criarmos as classes concretas com o operador **new**, utilizamos uma Fábrica Abstrata para criar os objetos em tempo de execução

- O código cliente não sabe qual classe concreta utilizamos

Fábrica Abstrata – 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.

Fábrica Abstrata – Estrutura



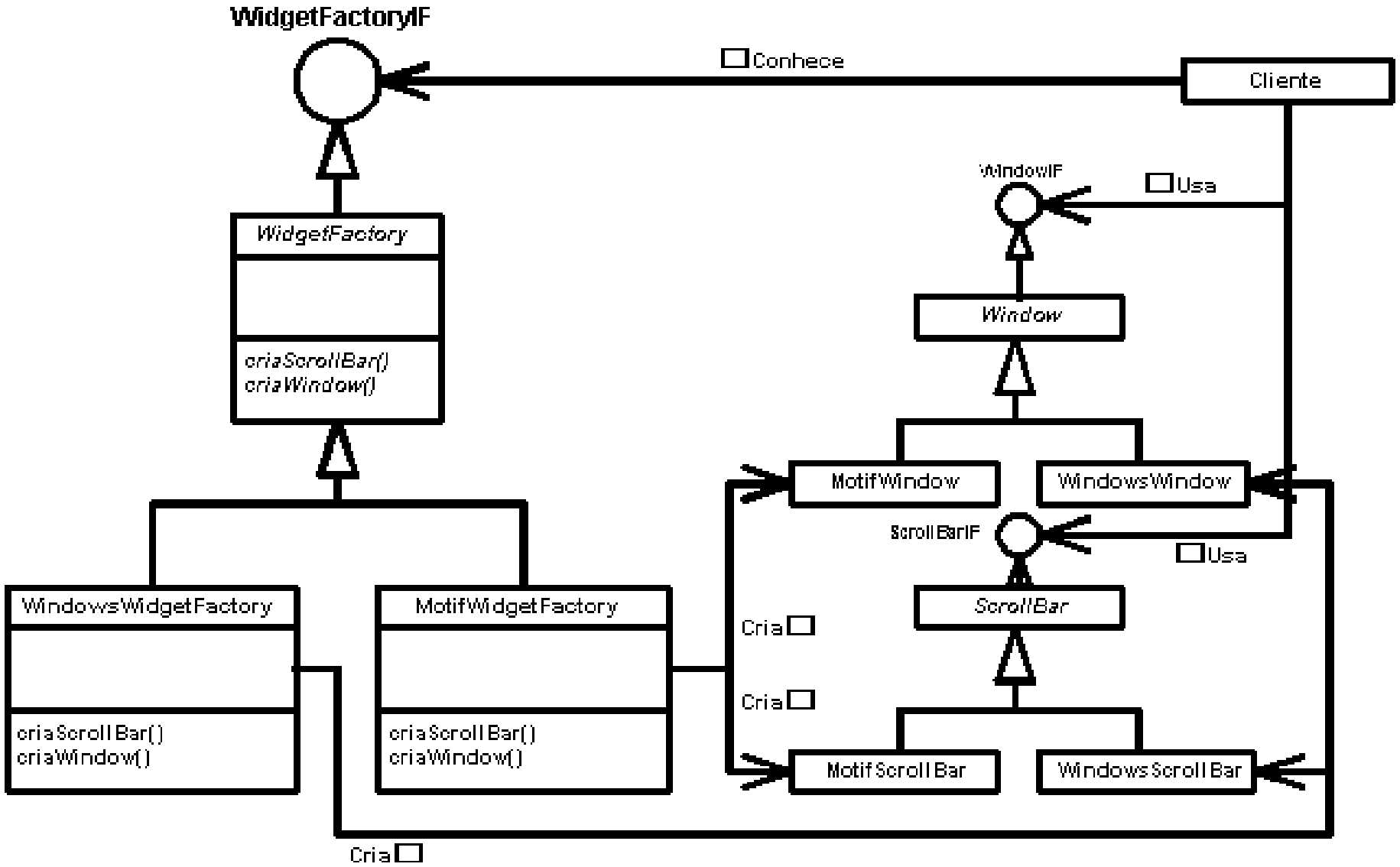
Fábrica Abstrata – Exemplo

- Para look-and-feel diferentes (Motif (Linux), Windows, Mac, Presentation Manager, etc.) temos formas diferentes de manipular janelas, scroll bars, menus, etc.
- Para criar uma aplicação com GUI que suporte qualquer look-and-feel, precisamos ter uma forma simples de criar objetos (relacionados) de uma mesma família
- Os objetos são dependentes porque não podemos criar uma janela estilo Windows e um menu estilo Motif (Linux)
- Java já resolveu este problema internamente no package awt usando Abstract Factory e você não precisa se preocupar com isso

Fábrica Abstrata – Exemplo

- Porém, você poderia estar usando C++ e precisaria cuidar disso você mesmo
- Uma classe (abstrata) (ou interface, em Java) "Abstract Factory" define uma interface para criar cada tipo de objeto básico (widgets no linguajar GUI)
- Também tem uma classe abstrata para cada tipo de widget (window, scroll bar, menu, ...)
- Há classes concretas para implementar cada widget em cada plataforma (look-and-feel)
- Clientes chamam a Abstract Factory para criar objetos
- Uma Concrete Factory cria os objetos concretos apropriados

Fábrica Abstrata – Exemplo



Fábrica Abstrata – 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 AbstractProduct

Fábrica Abstrata – 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**

Fábrica Abstrata – Conseqüências

- 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).

Fábrica Abstrata – 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**: 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()**).

Fábrica Abstrata – Exemplos de Código

- O GoF contém exemplos em C++ e Smalltalk
- Porém você encontrará exemplos em outras linguagens

Fábrica Abstrata – 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)

Fábrica Abstrata – Usos Conhecidos

- 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

Fábrica Abstrata – Padrões Relacionados

- Fábricas abstratas são normalmente implementadas com métodos fábrica (***FactoryMethod***) mas podem também ser implementados usando **Prototype**

- O uso de protótipos é particularmente importante em linguagens não dinâmicas como C++ e em linguagens "semi-dinâmicas" como Java

Fábrica Abstrata – Padrões Relacionados

- Em Smalltalk, não é tão relevante
  - Curiosidade: a linguagem **Self** não possui classes, toda criação de objetos é feita via clonagem

- Uma fábrica concreta é normalmente um **Singleton**

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 implementa-la duas vezes da mesma forma.”

- Talvez a última parte não seja sempre desejável.

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 implementa-la duas vezes da mesma forma.”

- Talvez a última parte não seja sempre desejável.

Saiba mais...

- Factory Méthod no Refactoring Guru (não deixe de ler): https://refactoring.guru/pt-br/design-patterns/abstract-factory

- Abstract Factory Teoria - Padrões de Projeto https://www.youtube.com/watch?v=UPSuHqNsNs4

- Exemplo de código em C++: https://www.bogotobogo.com/DesignPatterns/abstractfactorymethod.php