SCC0204 - Programação Orientada a Objetos

Design Patterns

Prof. Jose Fernando Rodrigues **Junior**
http://www.icmc.usp.br/~junio
junio@icmc.usp.br

INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
- USP -

Introdução

- **Soluções para problemas recorrentes do projeto (não do código)**
- **Elementos:**
  - **Nome do padrão**
    - Aumenta o vocabulário do projeto e o nível de abstração
  - **Problema**
    - São usados em contextos bem definidos
  - **Tipo**
    - Os padrões possuem categorização decorrente de sua características
  - **Consequências**
    - Prós e contras: espaço e tempo, reuso, complexidade

Introdução

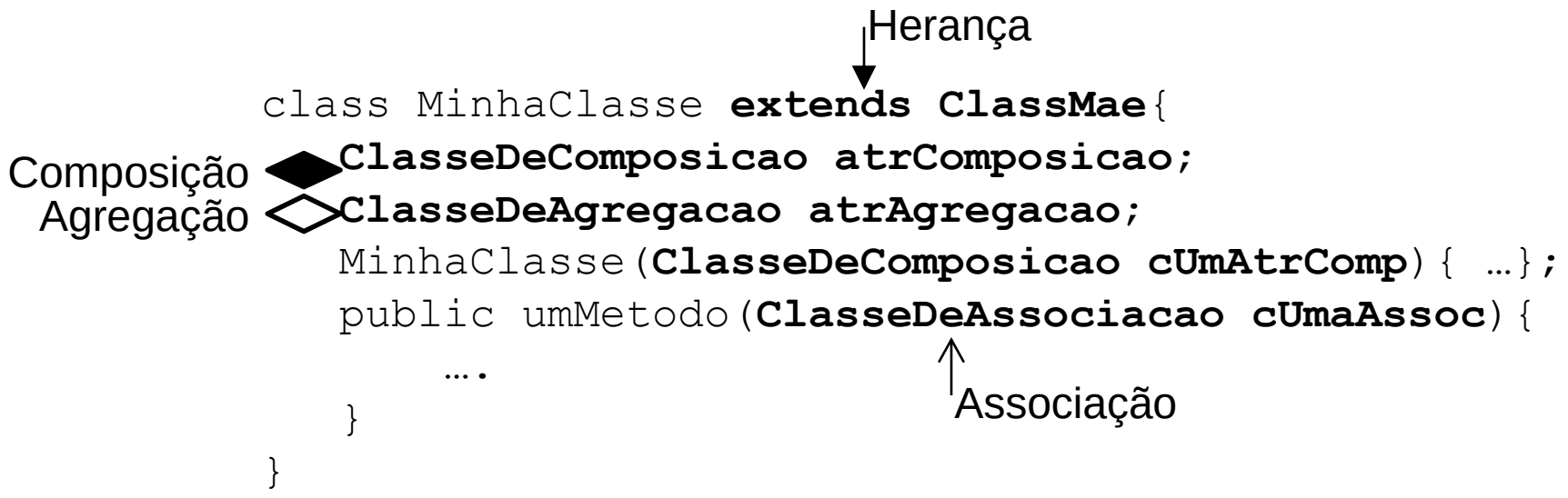
- **Desvantagem**
  - o uso de padrões requer que o desenvolvedor conheça **modelos** que nem sempre são evidentes no código → **engenharia de software**

- **Vantagens**
  - **menos código**
  - **mais abstração**
  - **reuso de design**

- Como o próprio nome diz, são **padrões**, portanto não são atrelados a situações específicas, ao invés disso podem ser usados em diferentes situações com características semelhantes

Introdução

- Os padrões de design combinam os recursos de POO de maneira engenhosa para alcançar características desejáveis:
  - Abstração
  - Flexibilidade
  - Simplicidade
  - Desacoplamento
- Breve revisão de UML:



Tipos e padrões mais conhecidos

- **Estrutural**
  - **Facade**
  - **Decorator (Aula 08)**
  - **Bridge**
  - **Proxy**
  - Adapter
  - Composite
  - Flyweight

- **Criacional**
  - **Singleton**
  - **Abstract Factory**
  - Factory Method
  - Builder
  - Prototype

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

Padrão Decorator (estrutural)

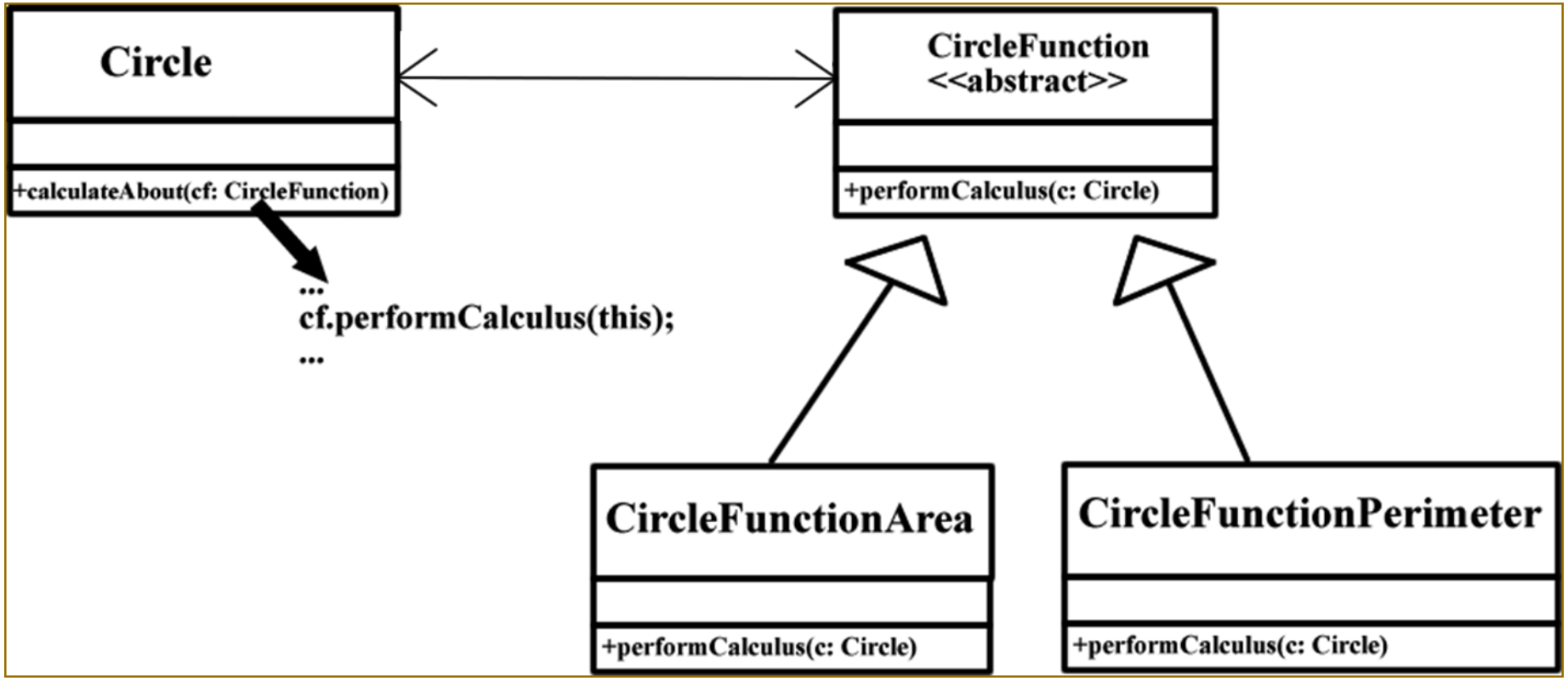
- O padrão decorator permite que **novas funcionalidades** sejam acrescentadas a uma classe **sem que seja necessário compilar** uma herança desta classe

- O maior exemplo do uso de decorator é a biblioteca de I/O do Java → Exemplo NetBeans

- Este padrão foi apresentado na Aula passada

Padrão Bridge (estrutural)

- Exemplo Netbeans
- Assim como o Decorator (Aula 08), o padrão Bridge permite que novas funcionalidades sejam acrescentadas a uma classe sem que seja necessário compilar uma herança desta classe



Padrão Bridge (estrutural)

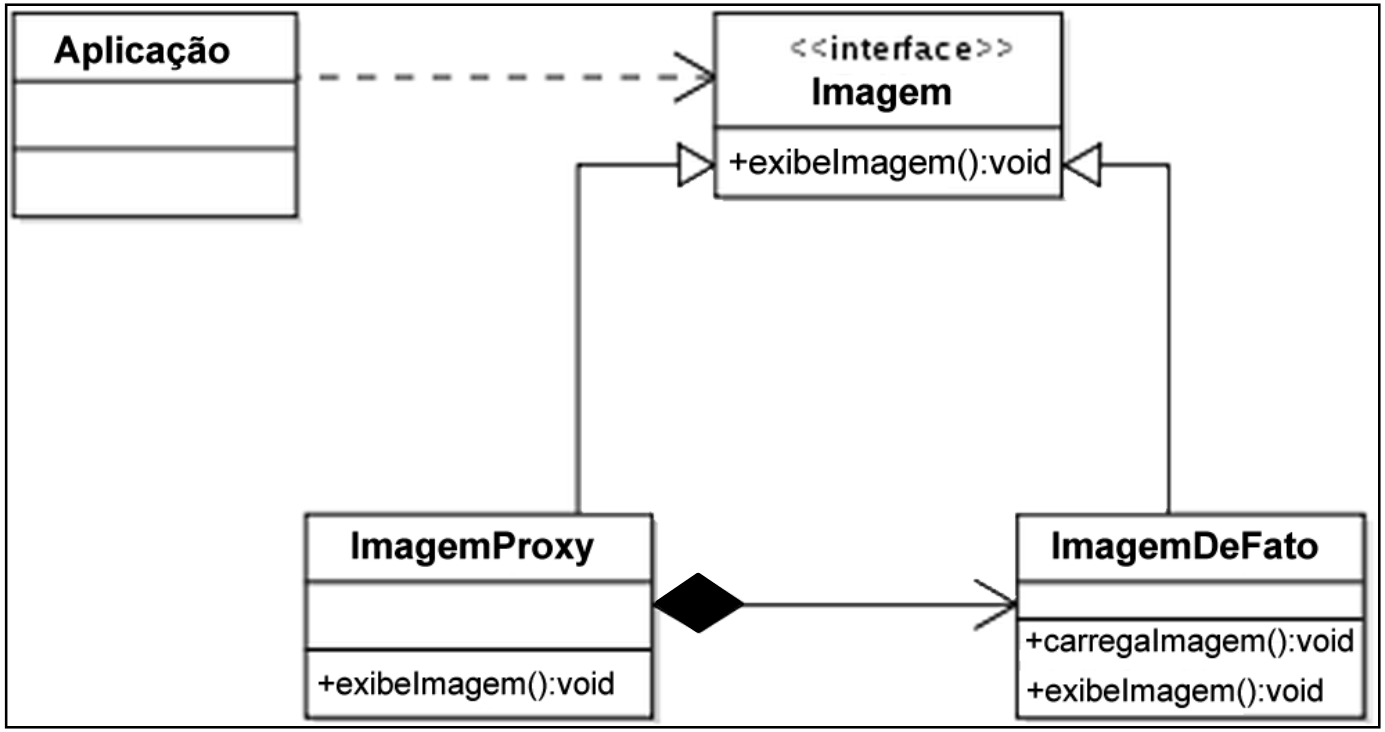
- Características:
  - **Separa abstração e implementação**
  - Permite **compartilhar implementações** entre múltiplos objetos
  - Aumenta a **capacidade de se estender o sistema**
  - **Esconde detalhes** de implementação

Padrão Bridge (estrutural)

- **Exercício**: a partir do exemplo Bridge, crie uma hierarquia a partir de uma classe abstrata Forma da qual descendem Circle e Square.
- Implemente duas classes para calcular a área e o perímetro do quadrado
- Exemplifique o uso deste código

Padrão Proxy (estrutural)

- Exemplo NetBeans

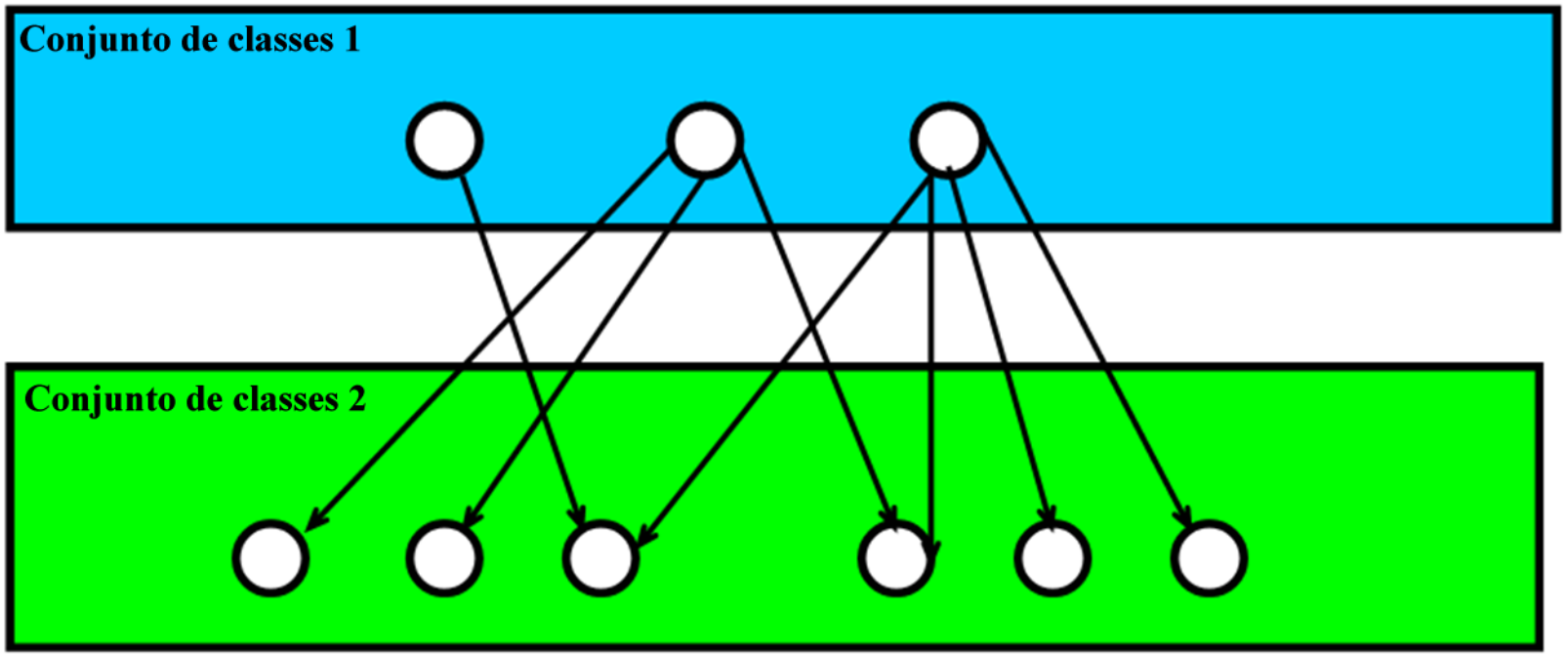


- Padrão usado para acrescentar controle a um objeto por meio de um outro objeto semelhante mas com comportamento diferente

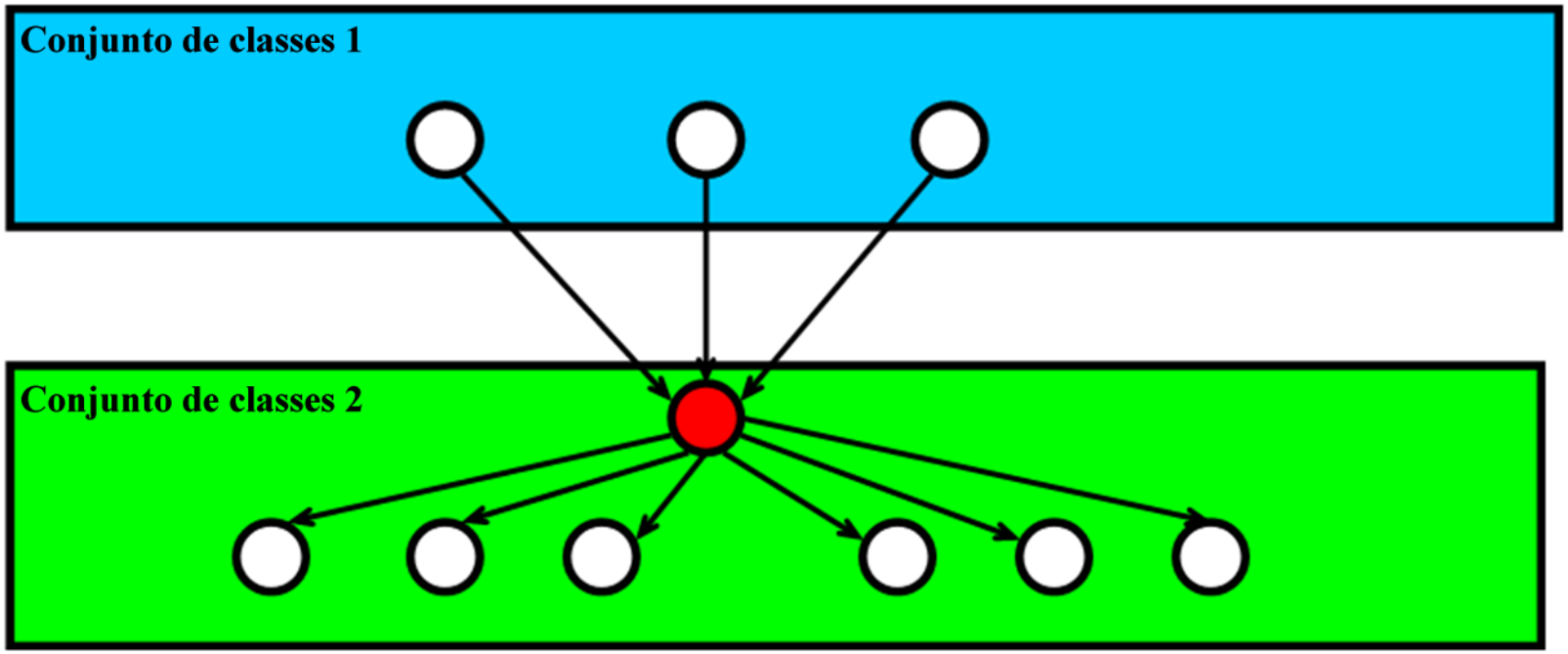
Padrão Proxy (estrutural)

- Características:
  - Permite que o **custo de se criar objetos seja gerenciado**
  - **Prorroga a instanciação** até que o objeto seja de fato usado
- Exemplos:
  - carregamentos de imagens
  - carregamento de recursos da rede

Padrão Facade (estrutural)

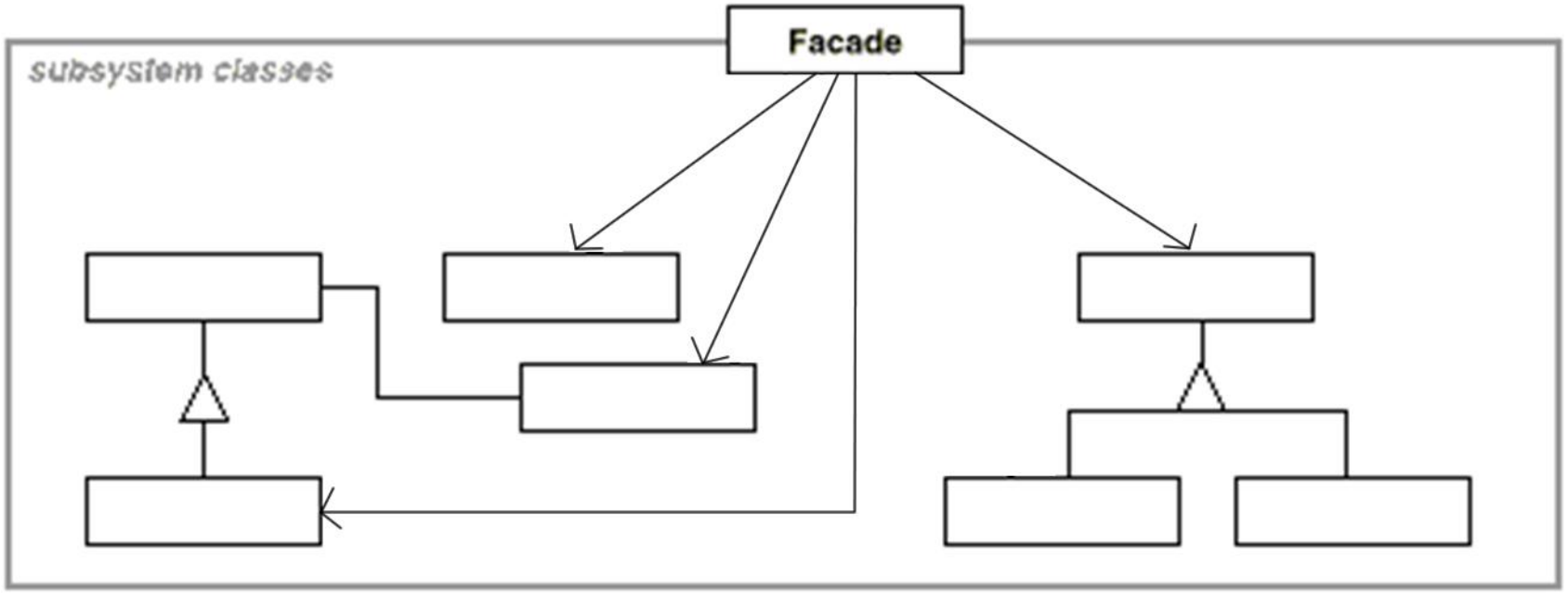


Padrão Facade (estrutural)



Padrão Facade (estrutural)

- Exemplo NetBeans



- Facade
  - Conhece quais classes do subsistema são responsáveis pelo atendimento de uma solicitação
  - Delega solicitações de clientes a objetos apropriados dos subsistemas
- Classes de subsistema
  - Implementam as funcionalidades do subsistema
  - Respondem a solicitações de serviços da Facade
  - Não têm conhecimento da Facade

Padrão Facade (estrutural)

- Características:
  - Reduz a **complexidade** do uso de sistemas
  - **Desacopla** subsistemas, reduz a dependência, e aumenta a portabilidade
  - Cria um **ponto de entrada para os subsistemas**
  - **Minimiza a comunicação** entre subsistemas
  - Tem potencial para **segurança e desempenho**
  - Poupa clientes da **complexidade de subsistemas**

Tipos e padrões mais conhecidos

- **Estrutural**
  - **Facade**
  - **Decorator**
  - **Bridge**
  - **Proxy**
  - Adapter
  - Composite
  - Flyweight

- **Criacional**
  - **Singleton**
  - **Abstract Factory**
  - Factory Method
  - Builder
  - Prototype

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

Padrão Singleton (criacional)

- Garante que apenas **uma única instância** de uma classe possa existir no projeto
- Se baseia em **um ponto de acesso global e único** à instância
- **Exemplos de aplicação**: conexão a banco de dados, gerenciador de janelas, gerenciador de arquivos, spooler de impressão, …

Padrão Singleton (criacional)

- Aplicação:

```
public class Singleton {
   private static Singleton instance = null;

   private Singleton() {
      // Existe apenas para evitar instanciação.
   }
   public static Singleton instance() {
      if(instance == null)
         instance = new Singleton();
      return instance;
   }
...
}
```

- Uso:

```
…
Singleton.instance().method();
…
```

Padrão Singleton (criacional)

- Aplicação:

O padrão Singleton é uma maneira **mais sistemática** (e aceita) para as funcionalidades conseguidas de outra maneira por meio de variáveis **globais**;

Evita que mais que uma instância de uma classe seja criada, e além disso **documenta** o código, avisando a quem vai usar que aquilo é único em todo o sistema.

-

```
Singleton.instance().method();
…
```

Padrão **Abstract** Factory (criacional)

- Exemplo NetBeans
- **Uma interface para a criação de famílias de objetos relacionados ou dependentes sem a necessidade de se especificar explicitamente quais são suas classes concretas.**

- Uma **técnica cujo objetivo é criar instâncias de outras classes**
- Ao invés de chamar o construtor de uma classe, **peça ao método Factory que crie o objeto para você**
- **Economia de código e redução de complexidade**

- Contexto:
  - suponha uma classe abstrata X, e um conjunto de classes XY, XZ, e XW descendentes de X; cada subclasse com características específicas;
  - o uso de XY, XZ ou XW depende de um dado problema, cujas características devem ser analisadas
  - sem o uso de factory, as características do problema devem ser analisadas toda vez que se for usar a hierarquia de X
  - com o uso de Factory, pode-se ter a definição e a instanciação da classe adequada centralizados em um único lugar, simplificando o uso posterior da hierarquia

Padrão **Abstract** Factory (criacional)

- Características:
  - Provê uma **biblioteca de objetos, expondo apenas uma interface simplificada**
  - **Cria famílias** de objetos relacionados
  - **Isola classes** concretas de suas super classes
  - **Traz independência** a respeito de como objetos são criados, compostos, e representados
  - Possibilita **definir restrições**
  - **Alternativa a Facade** para esconder classes específicas de plataforma
  - **Facilmente extensível** para um sistema ou família de classes

Tipos e padrões mais conhecidos

- **Estrutural**
  - **Facade**
  - **Decorator**
  - **Bridge**
  - **Proxy**
  - Adapter
  - Composite
  - Flyweight

- **Criacional**
  - **Singleton**
  - **Abstract Factory**
  - Factory Method
  - Builder
  - Prototype

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

Padrão Chain of Responsibility (comportamental)

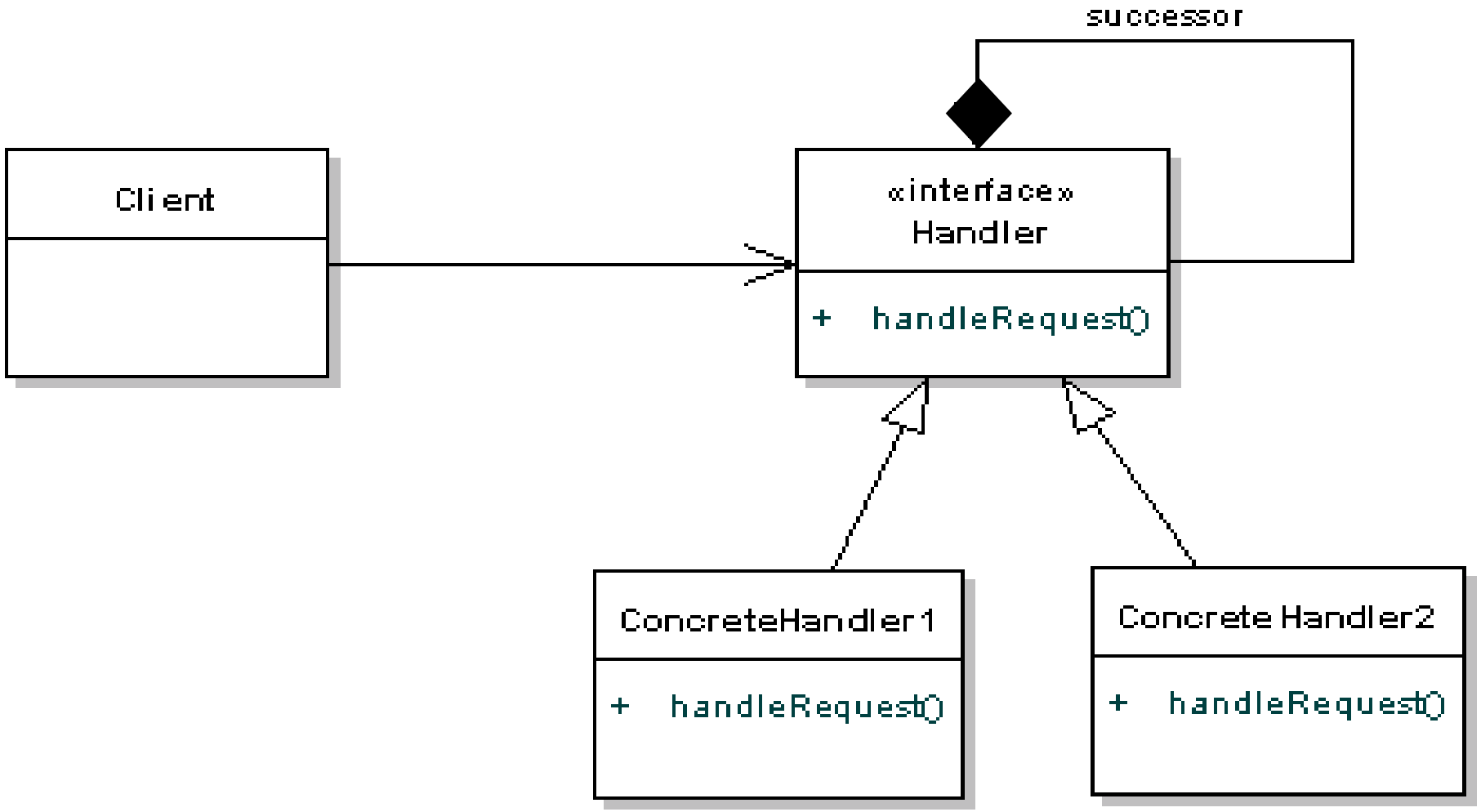
- Exemplo NetBeans

- Capaz de determinar **uma cadeia de processamento** ao longo de diferentes classes em uma hierarquia
- Exemplo:
  - Sistema de exceções
  - Processamento de arquivos

- Características:
  - Usada quando **não se sabe qual classe** deve tratar uma situação
  - **Flexibilidade** e redução de acoplamento

Padrão Chain of Responsibility (comportamental)

- Exemplo NetBeans



Padrão Command (comportamental)

- Exemplo NetBeans

- **Parametriza** o comportamento de uma classe
- **Popularmente conhecido como callback** em linguagens que aceitam ponteiros para funções

- Características:
  - **Usado em toda a API gráfica** de qualquer ambiente de desenvolvimento visual
  - Provê **forte generalização de código**

- Exemplo Jbutton: no exemplo do Jbutton, o padrão comando surge da definição da interface **ActionListener** que apenas define o método **actionPerformed**, o qual é implementado no **Jframe** recebendo o comando executado no evento do botão
- O método actionPerformed é disparado dentro de **AbstractButton. fireActionPerformed**

Padrão Mediator (comportamental)

- Exemplo NetBeans

- Provê **organização adicional** do código
- Define **interação entre classes com diferentes papéis** no Sistema
- Se **novas classes** precisarem entrar no Sistema, poucas adaptações serão necessárias

- Características:
  - **Despolui o código** de chamadas a configurações da interface gráfica
  - **Centraliza a manipulação de operações relacionadas**
  - Provê **desacoplagem**

Padrão Mediator (comportamental)

- Exemplo NetBeans
- Prov
- Defi Sistema
- Se poucas adap
- Cara
  - **D** interface g
  - **C**
  - F

O mediador difere do Facade pois no Facade os objetos envolvidos não tem conhecimento do Facade, que é usado para se obter as funcionalidades.

Já o Mediator é conhecido por todos os objetos envolvidos, e não é usado diretamente para se obter as funcionalidades, as quais são dadas pelos próprios objetos e não por uma fachada.

Padrão Observer (comportamental)

- Exemplo NetBeans

- É composto por objetos **observadores** e objetos **observáveis** (ou observados)
- Seu uso é motivado por operações de **atualização** ocorridas nos objetos observáveis
- Por exemplo: imagine uma **planilha de dados** e um **gráfico** correspondente – toda vez que os dados da planilha forem alterados, o gráfico deverá ser informado e ser atualizado; neste exemplo, a planilha é o objeto observável e o gráfico é o objeto observador
- Vantagem: **desacoplamento** entre o objeto observável e seus observadores – para se ter uma observação, basta que um observador se registre ao objeto observável

- A API Java oferece funcionalidades **Observable** e de **Observer**

Padrão Observer (comportamental)

- **Os objetos observados (Observable) possuem uma lista de objetos observadores (Observer)**

- Quando ocorrem **atualizações nos objetos observados**, eles devem **notificar explicitamente a cada um dos observadores** de sua lista (notifyObservers)

- Cada observador então **executa um método de update**, o qual processa os dados dos objetos que ele observa

Tipos e padrões mais conhecidos

- **Estrutural**
  - **Facade**
  - **Decorator**
  - **Bridge**
  - **Proxy**
  - Adapter
  - Composite
  - Flyweight

- **Criacional**
  - **Singleton**
  - **Abstract Factory**
  - Factory Method
  - Builder
  - Prototype

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

Problemas com o Catálogo de Padrões

- Armazenamento, busca e manutenção da **documentação** de padrões
- Projetistas usando novos padrões precisam considerar onde o seu novo padrão **se encaixa** no projeto
- **Publicação** de padrões disponíveis
- Todos os **usuários precisam ser atualizados** sobre o conteúdo do catálogo

Perigos do Uso de Padrões

- O uso de padrões de uma maneira descontrolada pode originar **projetos sobrecarregados**

- Desenvolvedores precisam de **tempo** para entender os catálogos de padrões relevantes
  - Precisam de fácil acesso a catálogos relevantes
  - Precisam ser **treinados** no uso de padrões

- O uso de padrões não garante bons projetos, para usar um padrão, siga três passos:

  1) Entenda seu problema

  2) Entenda o padrão de projeto

  3) Entenda como o padrão resolve seu problema

Livros sobre Padrões de Software

Categoria do Padrão	Título	Autores / Editores
Proposição original (Gang of Four)	Design Patterns: Elements of Reusable Object-Oriented Software (the GangOfFour)	ErichGamma, RichardHelm, RalphJohnson, and JohnVlissides
Geral	Head First Design Patterns, O'Reilly	Freeman
Análise OO	Analysis Patterns: Reusable Object Models	Martin Fowler
Arquitetura	Pattern-Oriented Software Architecture: A System of Patterns	Buschmann et al.
Projeto	Design Patterns: Elements of Reusable Object-Oriented Software	Gamma et al.
	Anti-Patterns: Refactoring Software, Architectures, and Projects in Crisis	William J. Brown et al.
	Design Patterns Java™ Workbook	Steven John Metsker

- Exemplos: http://www.javacamp.org/designPattern/