Análise e Projeto de Sistemas

Antônio da Mota Moura Júnior
jmoura.unibh@gmail.com

“Princípios de Análise e Projeto de Sistemas” - Eduardo Bezerra (Cap.8).
“Padrões de Projeto (Design Patterns)- soluções reutilizáveis de software orientado a objetos”- Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides.

Padrões de Projeto (Design Patterns)

- É da natureza do desenvolvimento de software o fato de que os mesmos problemas tendem a acontecer diversas vezes.
- Um ***padrão de projeto*** corresponde a um esboço de uma solução reusável para um problema comumente encontrado em um contexto particular.
- Estudar esses padrões é uma maneira efetiva de aprender com a experiência de outros.
- O texto clássico sobre o assunto é o de Erich Gamma et al.
  - Esses autores são conhecidos *Gang of Four*.
  - Nesse livro, os autores catalogaram 23 padrões.

Padrões GoF

- Os padrões GoF foram divididos em três categorias:
  - ***Criacionais***: procuram separar a operação de uma aplicação de como os seus objetos são criados.
  - ***Estruturais***: provêem generalidade para que a estrutura da solução possa ser estendida no futuro.
  - ***Comportamentais***: utilizam herança para distribuir o comportamento entre subclasses, ou agregação e composição para construir comportamento complexo a partir de componentes mais simples.

Padrões GoF

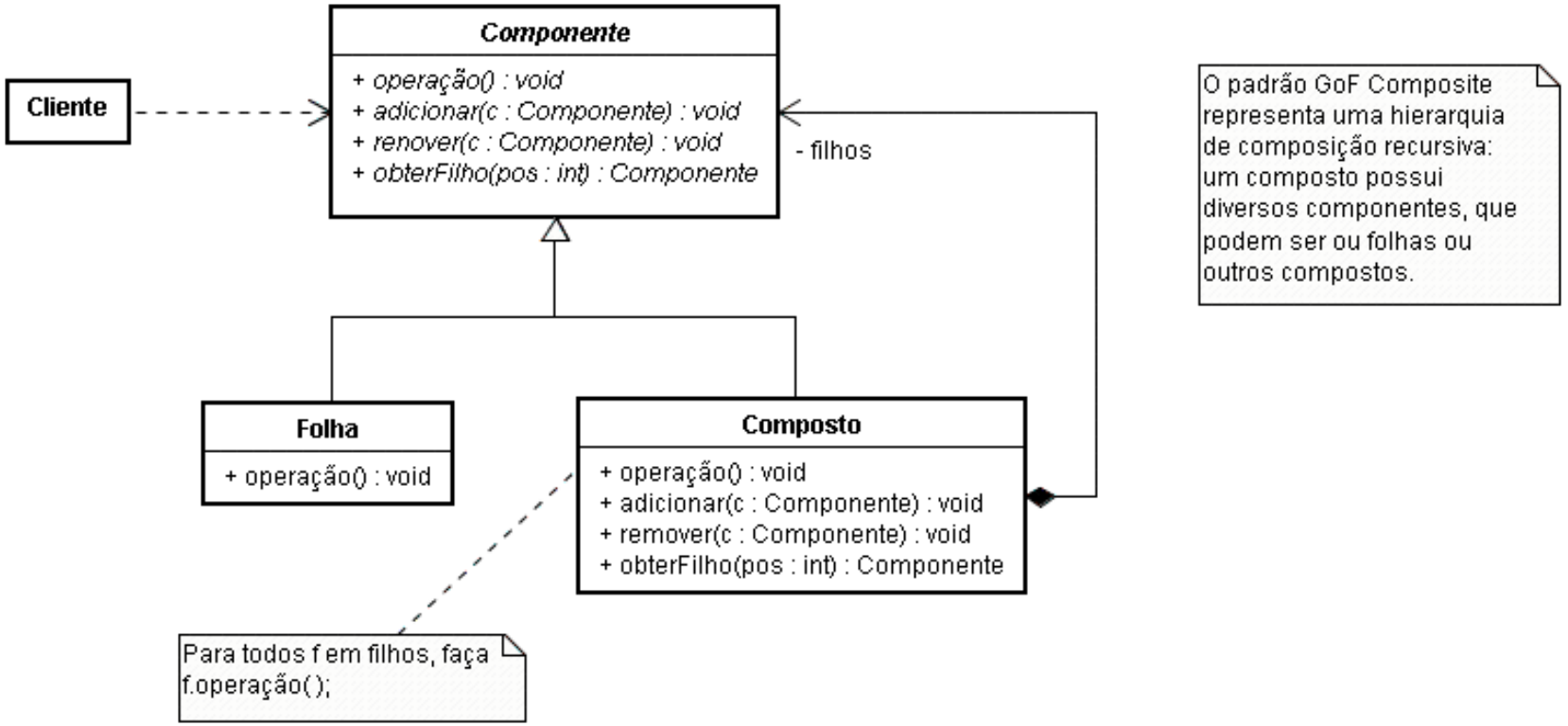
Criacionais	**Estruturais**	**Comportamentais**
Abstract Factory Builder **Factory Method** Prototype Singleton	Adapter Bridge **Composite** Decorator **Façade** Flyweight Proxy	Chain of Responsibility Command Interpreter Iterator **Mediator** Memento **Observer** State **Strategy** Template Method Visitor

Composite

- Categoria: Padrões Estruturais
- Problema: como definir uma relação hierárquica entre objetos de tal forma que tanto o objeto todo quanto os objetos parte sejam equivalentes? (E. Bezerra)
- Intenção: compor objetos em estruturas de árvores para representarem hierarquias todo-partes. Composite permite aos clientes tratarem de maneira uniforme objetos individuais e composições de objetos. (E.Gamma et ali)

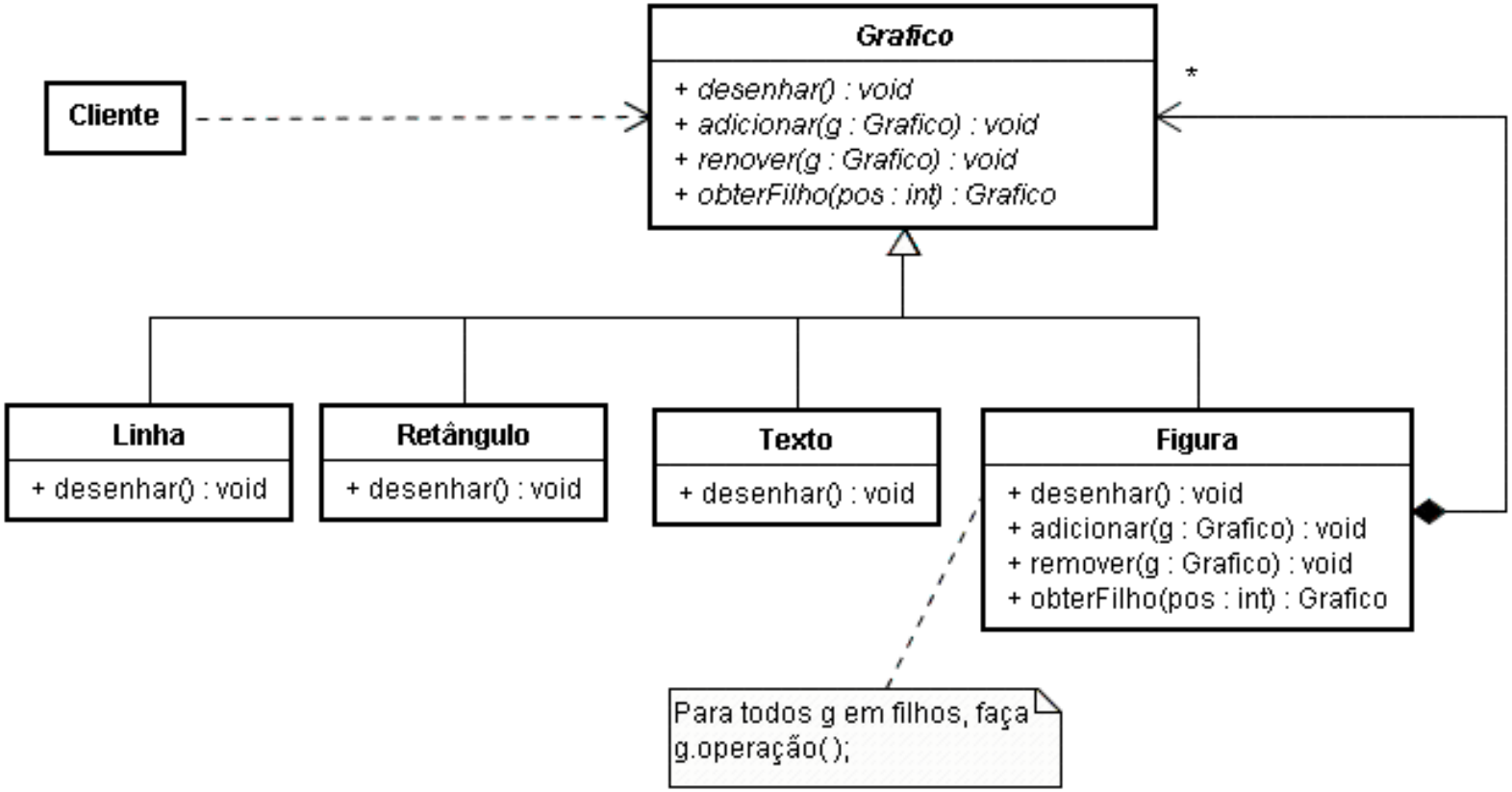
Composite

- Estrutura



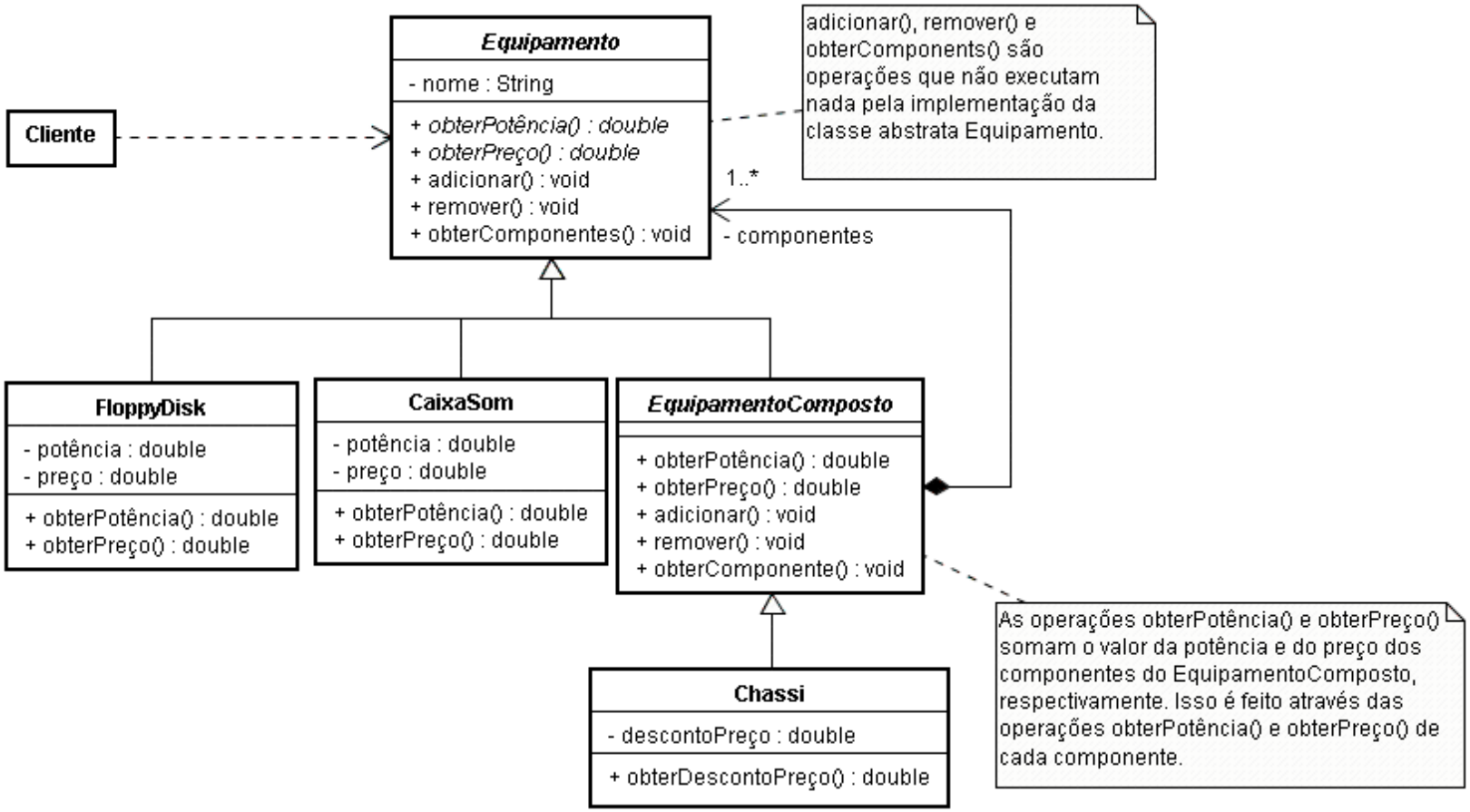
Composite

- Exemplo1



Composite

- Exemplo2 (Sistema de som estereofônico)



Composite

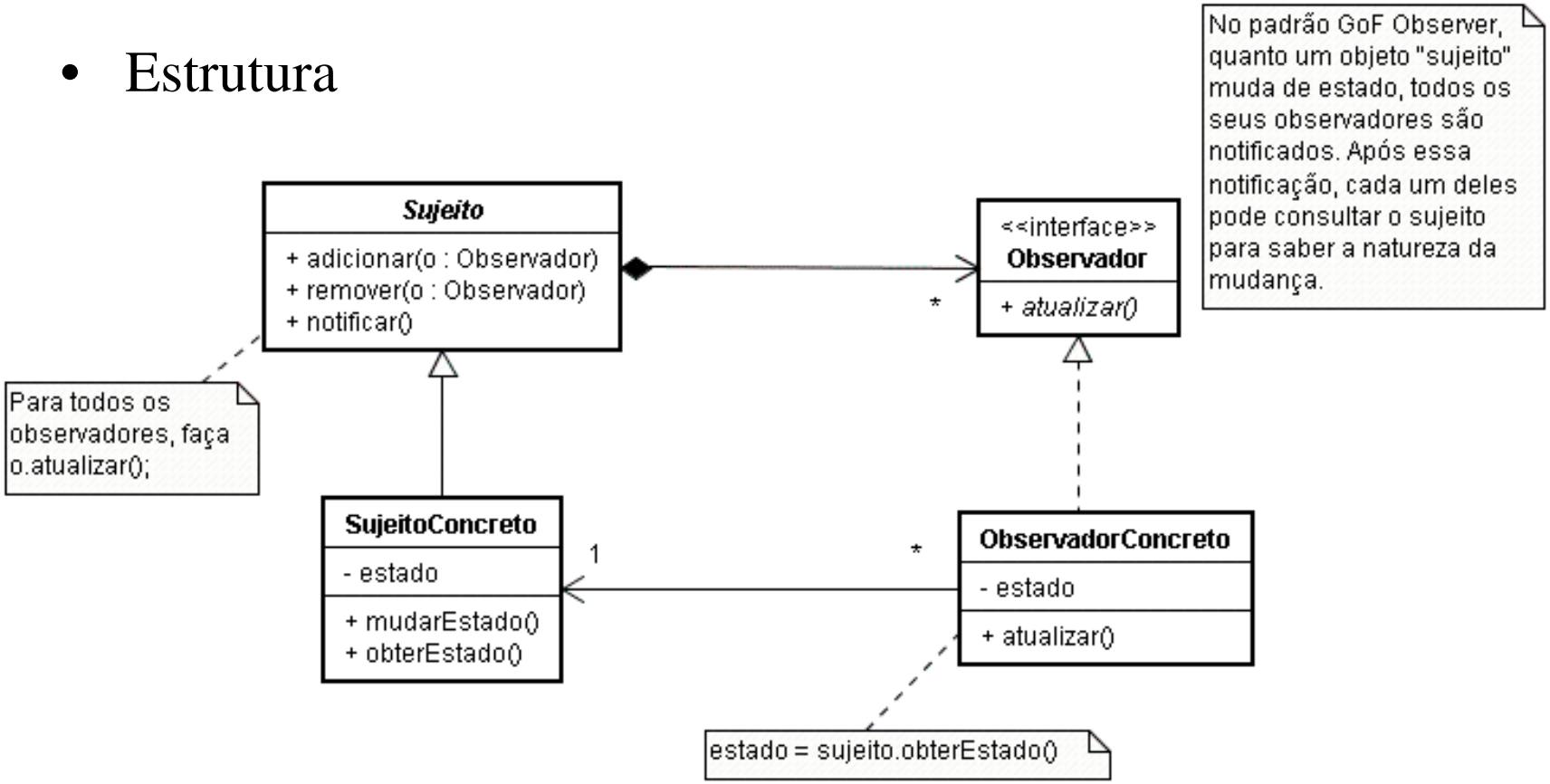
- Algumas observações sobre a implementação do padrão:
  - Referência explícitas aos pais: manter referências dos componentes-filhos para seus pais pode simplificar o percurso e a administração de uma estrutura composta. Simplifica mover-se para cima na estrutura e excluir um componente.
  - Compartilhamento de componentes: é útil compartilhar componentes, por exemplo, para reduzir os requisitos de armazenamento, quando um componente pode ter mais que um pai.
  - Declarando as operações de gerenciamento de filhos (adicionar, remover etc.): Essas operações devem ser declaradas na classe Componente e fazer com que tenham sentido para as classes Folha. Uma opção é criar uma operação concreta "public Composto getComposto()" na classe abstrata Componente, cuja implementação retorne nulo. Com isso, os clientes de Componente saberão quando o objeto é uma Folha ou um Composto.

Observer

- Categoria: Padrões Comportamentais
- Problema: é útil quando há um objeto central e diversos outros objetos *dependem* do primeiro. Se houver uma modificação no estado do objeto central, os objetos dependentes devem ser notificados.
- Intenção: definir uma dependência um-para-muitos entre objetos, de maneira que quando um objeto muda de estado todos os seus dependentes são notificados e atualizados automaticamente (E.Gamma et ali). Desejamos que haja um acoplamento fraco entre objetos dependentes e objeto central, pois isso melhor a reusabilidade.

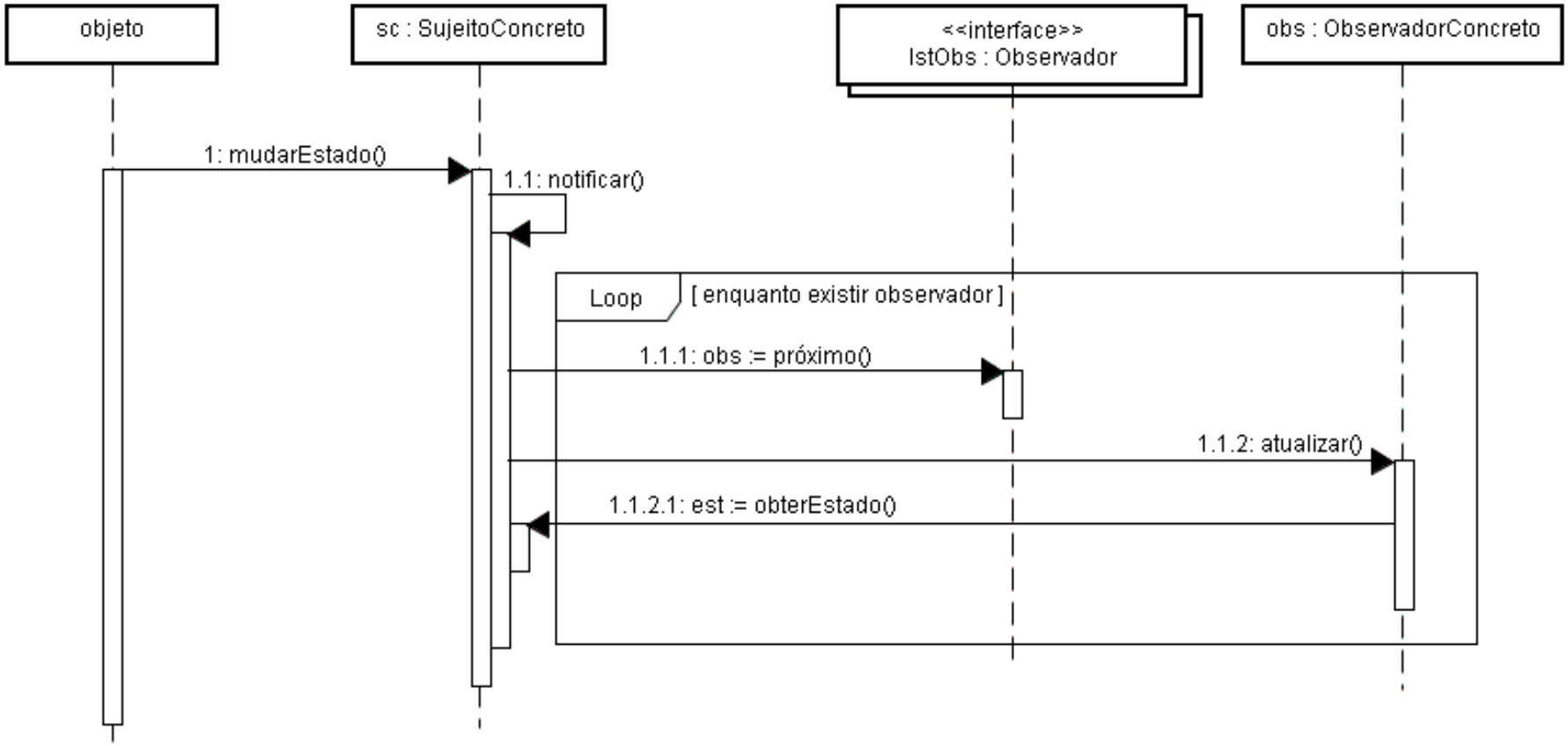
Observer

- Estrutura



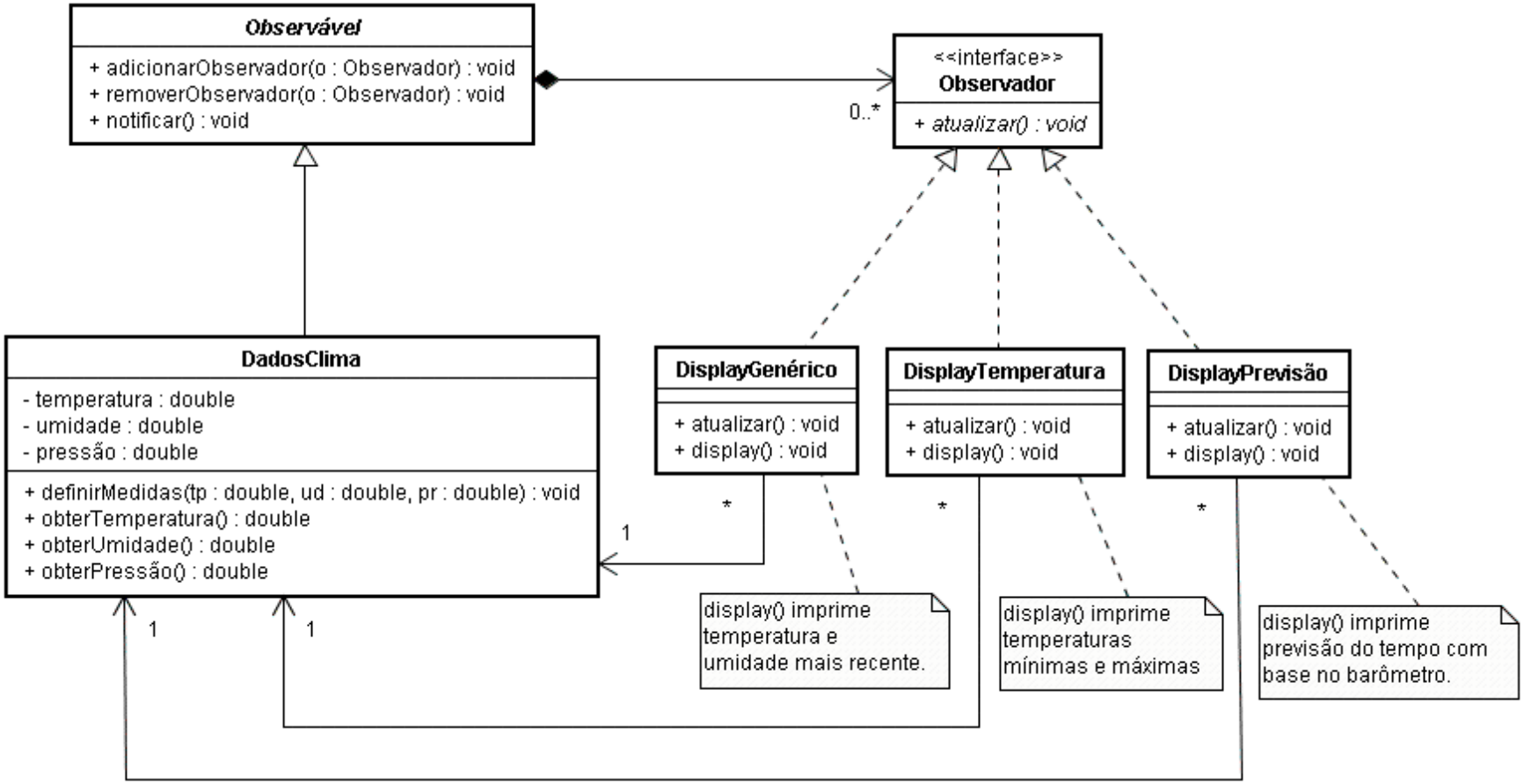
Observer

- Exemplo de colaboração entre objetos



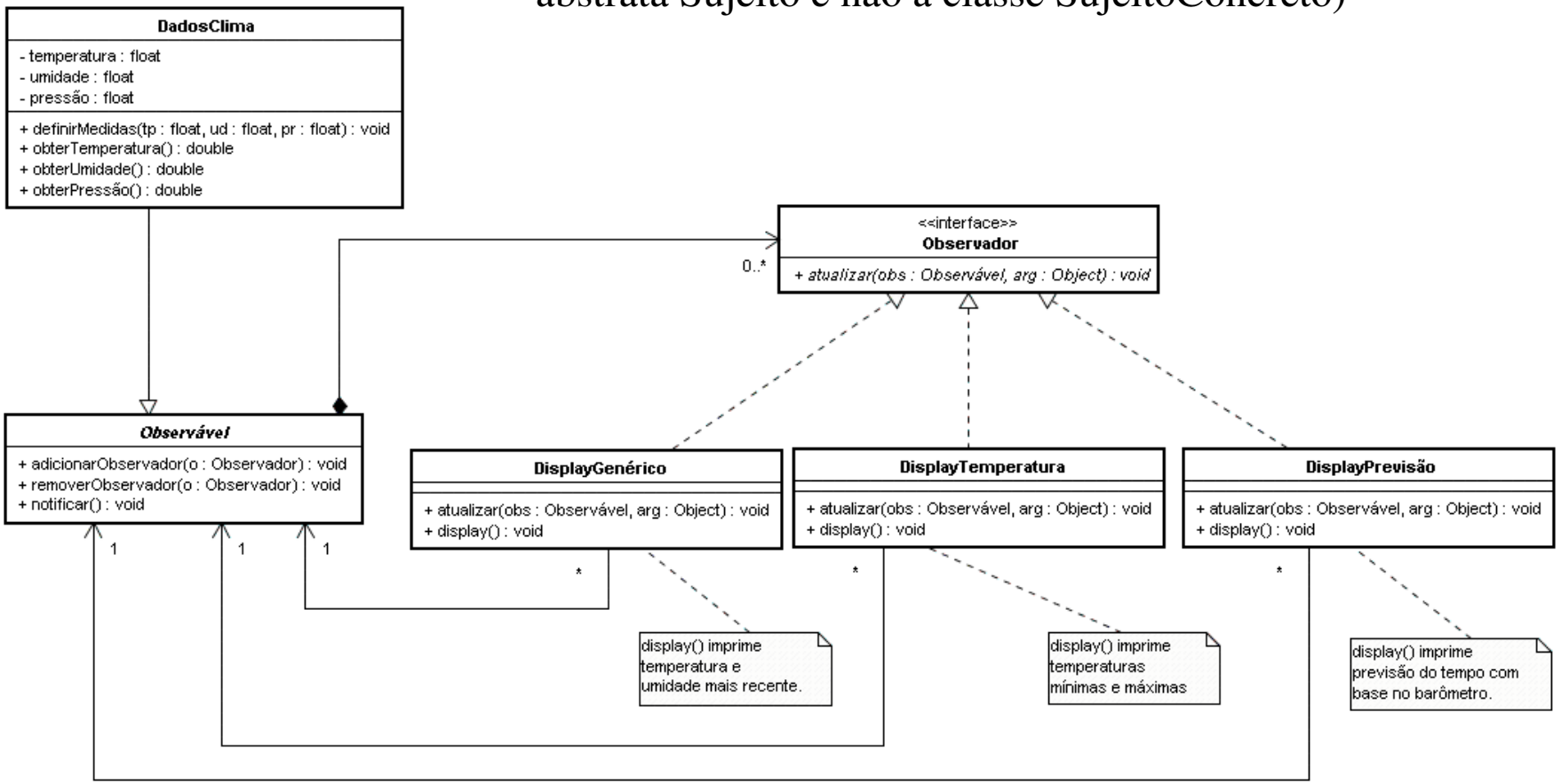
Observer

- Exemplo (Estação meteorológica)



Observer

- Exemplo (Estação meteorológica - ObservadorConcreto está ligado à classe abstrata Sujeito e não à classe SujeitoConcreto)



Observer

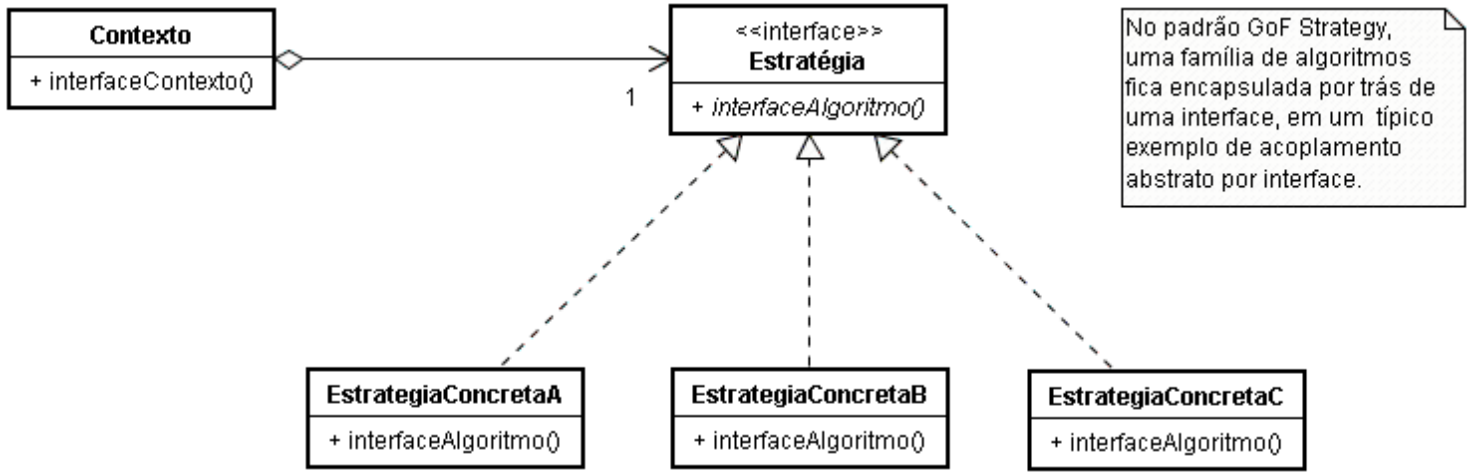
- Algumas observações sobre a implementação do padrão:
  - O construtor do ObservadorConcreto recebe um objeto Sujeito.
  - A classe abstrata Sujeito (ou Observável, ou Observable) fornece duas funcionalidades importantes para suas subclasses: a de notificação dos dependentes; e a de manutenção desses dependentes. Essas operações são definidas em termos da interface Observador (ou Observer).
  - Aplicação prática: o padrão Observer é útil para permitir que uma camada de software mais genérica possa enviar sinais para uma camada menos genérica. O elemento da camada mais genérica, representado pelo objeto central, pode estar associado a diversos objetos que dele dependem e que estão localizados na camada mais específica, que são representados por objetos dependentes.

Strategy

- Categoria: Padrões Comportamentais
- Problema: Necessidade de utilizar, em momentos diferentes, diferentes algoritmos para realizar uma tarefa computacional.
- Intenção: definir uma família de algoritmos, encapsular cada um deles por trás de uma interface e torná-los intercambiáveis nos clientes. Strategy permite que o algoritmo varie independentemente dos clientes que o utiliza. Pode ser interpretado com uma forma de desacoplar uma região de código cliente de uma tarefa das diferentes maneiras de implementar essa tarefa.

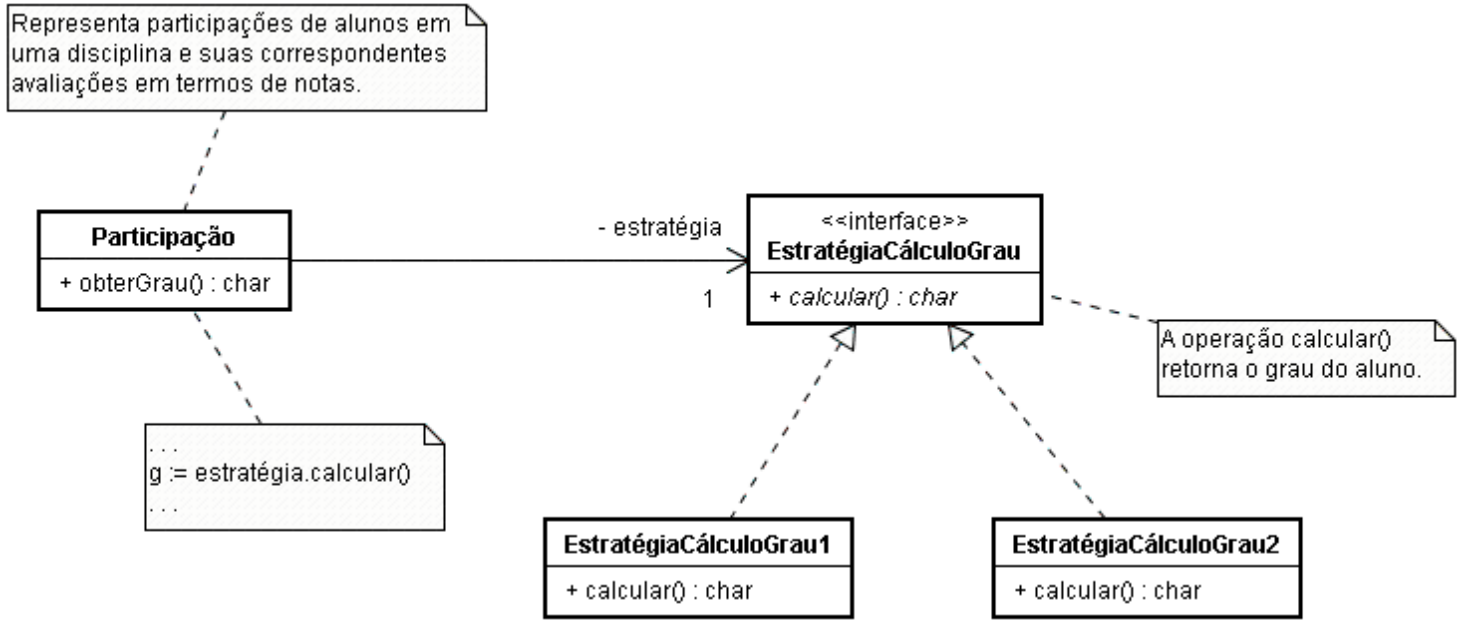
Strategy

- Estrutura



Strategy

- Exemplo (Sistema de Controle Acadêmico - cálculo do grau final de um aluno)



Strategy

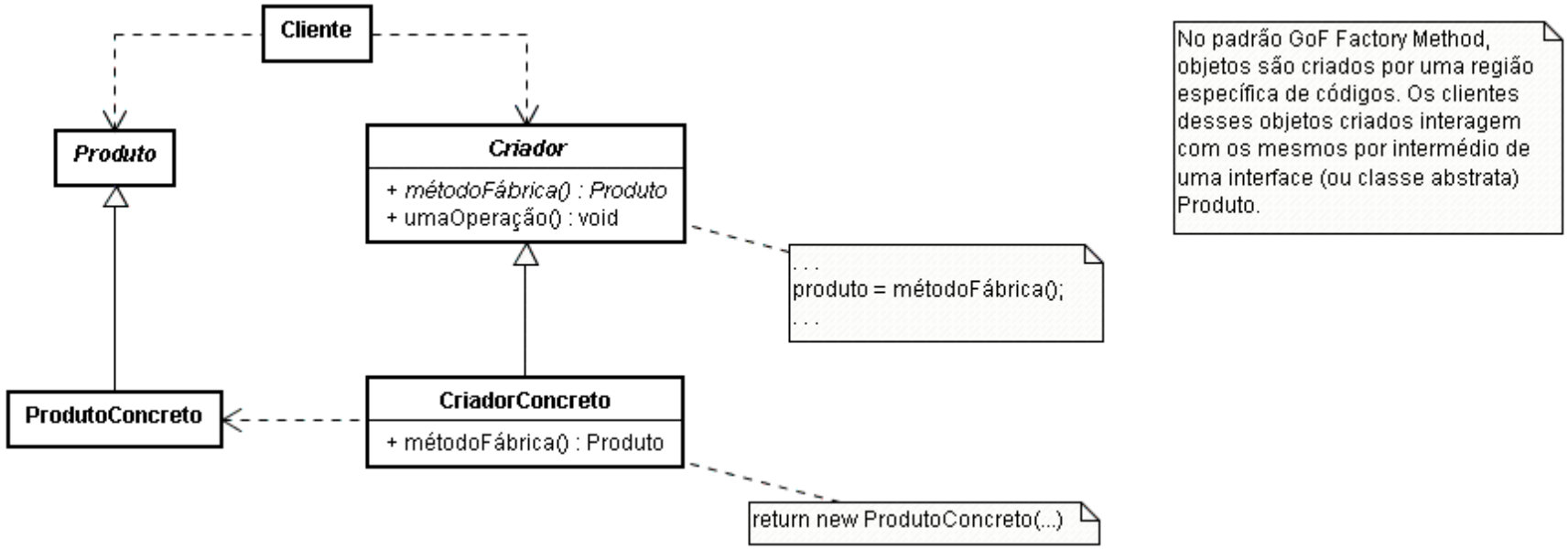
- Algumas observações sobre a implementação do padrão:
  - uma região de código da aplicação (Contexto) fica responsável por identificar qual é a estratégia vigente para cálculo de grau e por instanciar o objeto da subclasse correspondente;
  - grande vantagem está na flexibilidade resultante;
  - Estratégia pode ser implementada como uma interface ou como uma classe abstrata. Em ambos os casos, entretanto, o acoplamento abstrato é mantido entre os elementos envolvidos;
  - o padrão tem uma deficiência potencial no fato de que um cliente deve compreender como Estratégias diferem, antes que ele possa selecionar a mais apropriada. Os clientes podem ser expostos a detalhes e aspectos de implementação. Portanto, você deveria usar o padrão Strategy somente quando a variação em comportamento é relevante para os clientes.

Factory Method

- Categoria: Padrões Criacionais
- Problema: a operação de instanciação de um objeto pode ser bastante complexa. Além disso, pode ser que não seja adequado fazer com que uma região de código que necessite de um serviço tenha uma referência direta para a classe que o fornece. Uma razão para isso pode ser o fato de que a forma de implementar tal serviço é instável, no sentido de precisar ser alterada no futuro. Se a região de código instanciar diretamente a classe que lhe fornece determinado serviço, essa região fica definitivamente acoplada a essa classe fornecedora e a sua forma específica de prover o serviço requerido.
- Intenção: definir uma interface para instanciar objetos, mas deixar as subclasses decidirem que classe instanciar. O Factory Method permite adiar a instanciação para subclasses.

Factory Method

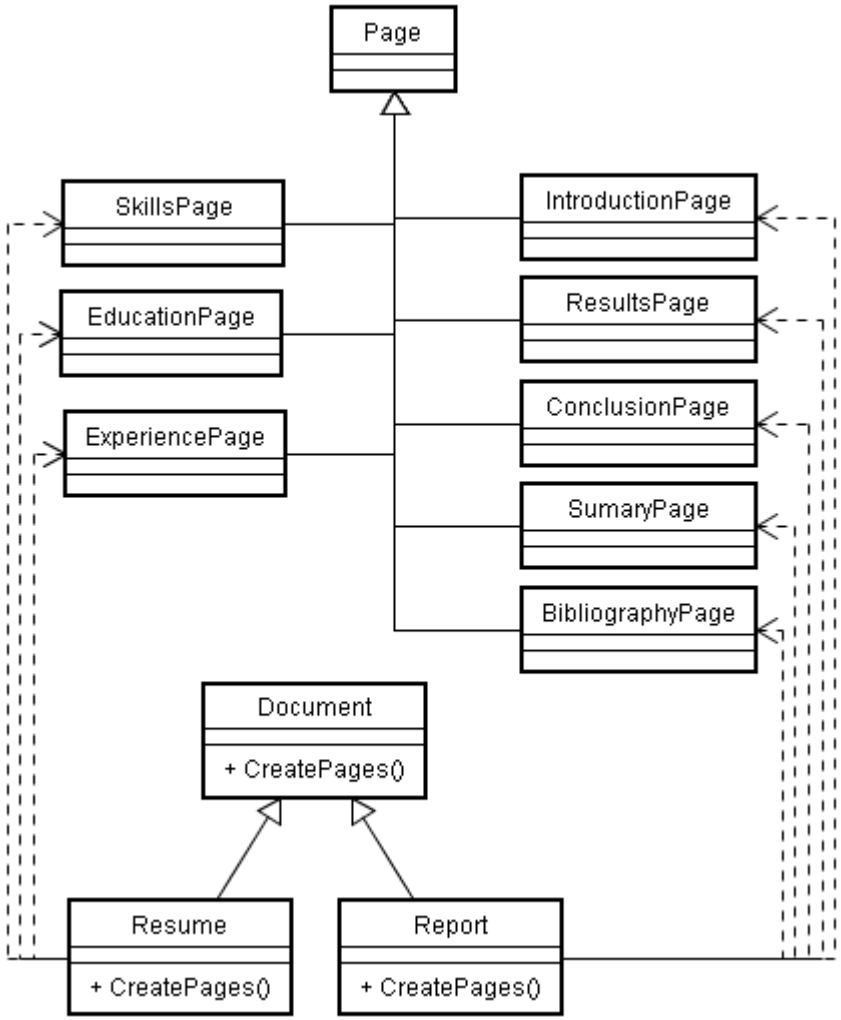
- Estrutura



Factory Method

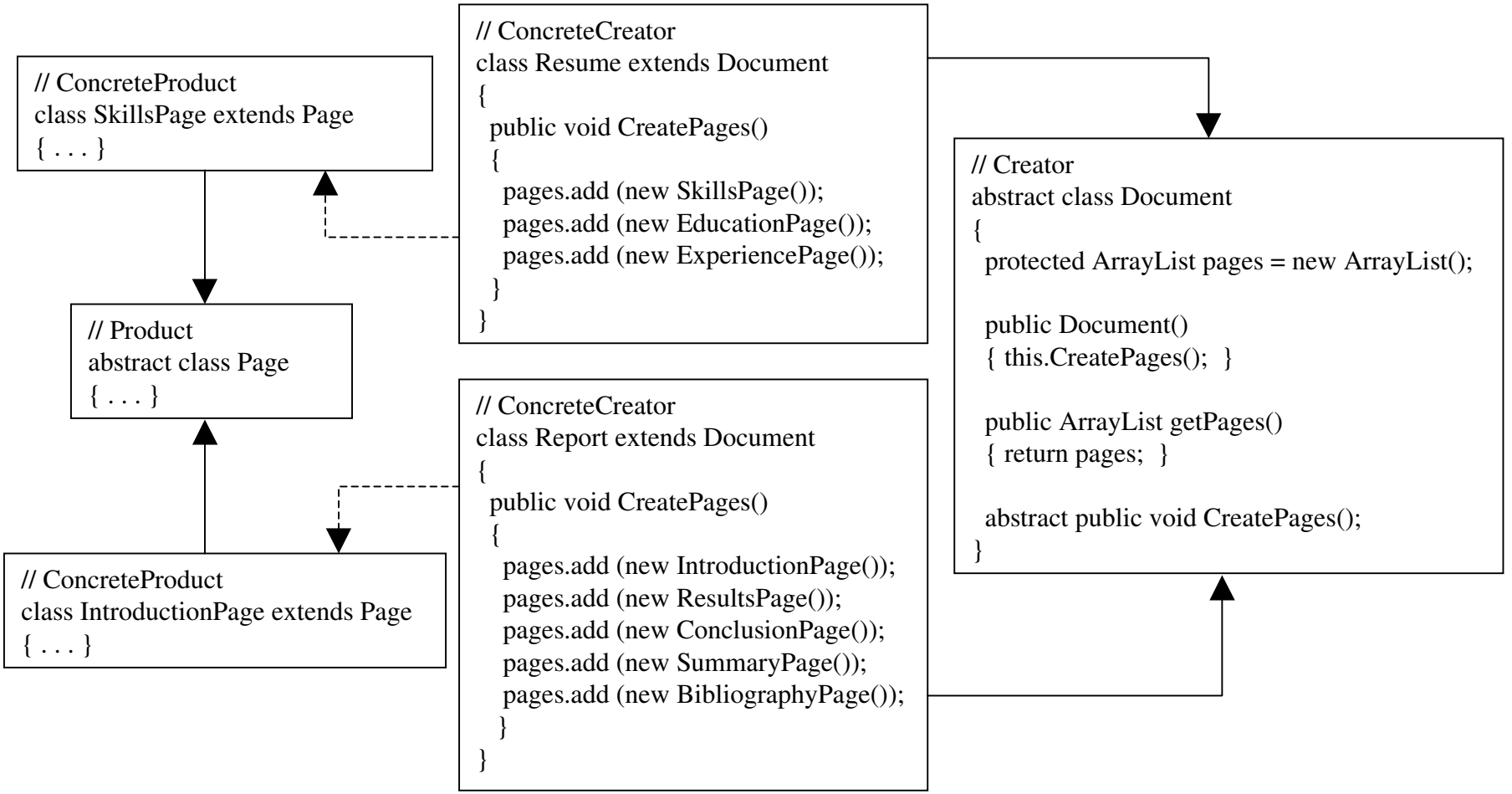
- Exemplo

Este exemplo de código concreto, demonstra o FactoryMethod sendo utilizado para oferecer flexibilidade na criação de documentos diferentes. As classes Report e Resume, subclasses de Document criam versões especializadas da classe Document. Neste caso, o FactoryMethod é chamado no método construtor da classe base (Document).
(Fonte:
http://www.oodesign.com.br/patterns/FactoryMethod.html)



Factory Method

- Código Exemplo



Factory Method

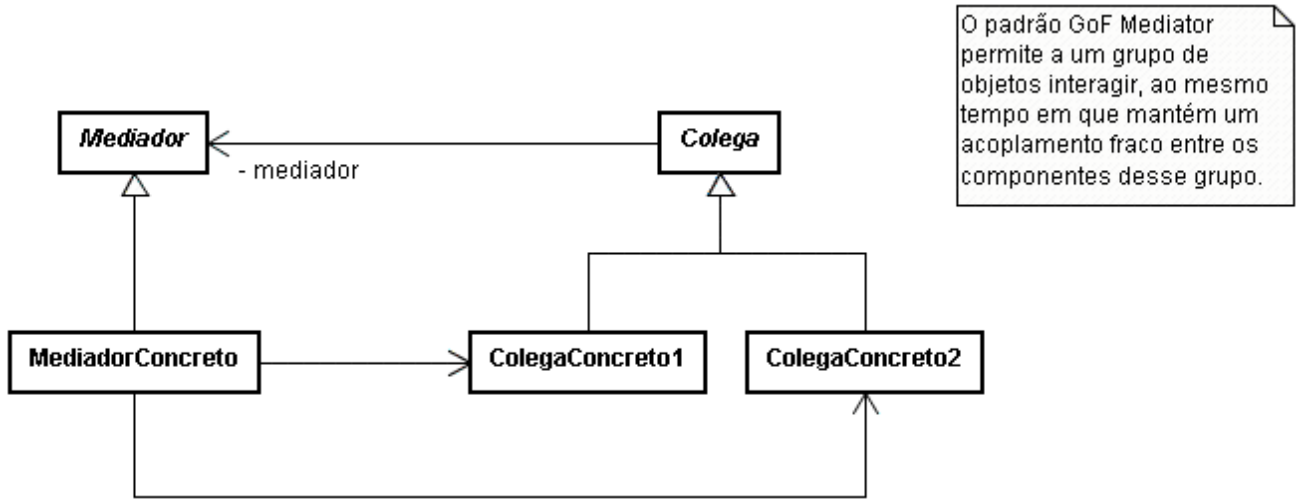
- Algumas observações sobre a implementação do padrão:
  - a fábrica de objetos precisa fazer referência às classes concretas cujos objetos fornecem o serviço requerido (acoplamento concreto); entretanto, essa referência fica localizada em uma região de código particular (na fábrica de objetos) e é consequentemente aceitável;
  - quaisquer mudanças nas classes que fornecem o serviço em questão ficam localizadas na fábrica de objetos e não afetam os clientes;
  - sua utilização adiciona complexidade ao projeto, como a maioria dos padrões de projeto: adicionam *flexibilidade* à custa de adicionarem *complexidade* à solução;
  - considerar se a classe que fornece o serviço é realmente suscetível a mudanças futuras, do contrário, a referência (instanciação) direta da classe que fornece o serviço (em vez de utilização do padrão Factory Method) é uma alternativa mais viável;
  - o padrão GoF **Abstract Factory** é uma extensão do Factory Method para criação de família de objetos relacinados.

Mediator

- Categoria: Padrões Comportamentais
- Problema: o projeto orientado a objetos encoraja a distribuição de comportamento entre vários objetos. Tal distribuição pode resultar em uma estrutura de objetos com muitas conexões entre eles; na pior situação, cada objeto acaba tendo conhecimento sobre todos os outros objetos.
- Intenção: definir um objeto que encapsula a forma como um conjunto de objetos interage. O Mediador promove o acoplamento fraco ao evitar que os objetos se refiram uns aos outros explicitamente e permite variar suas interações independentemente.

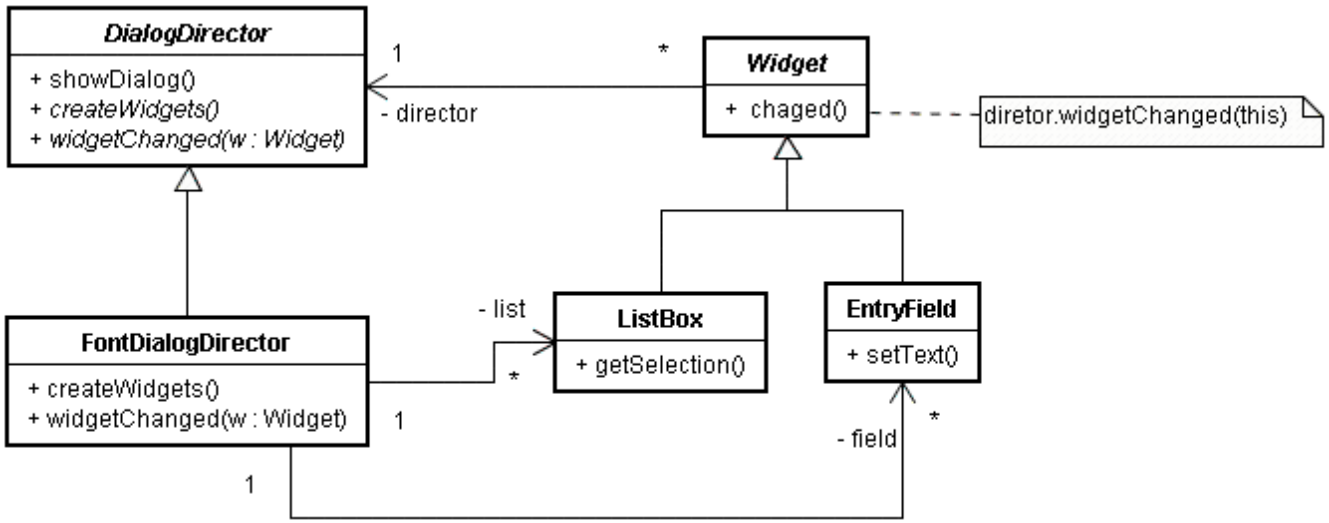
Mediator

- Estrutura



Mediator

- Exemplo



Mediator

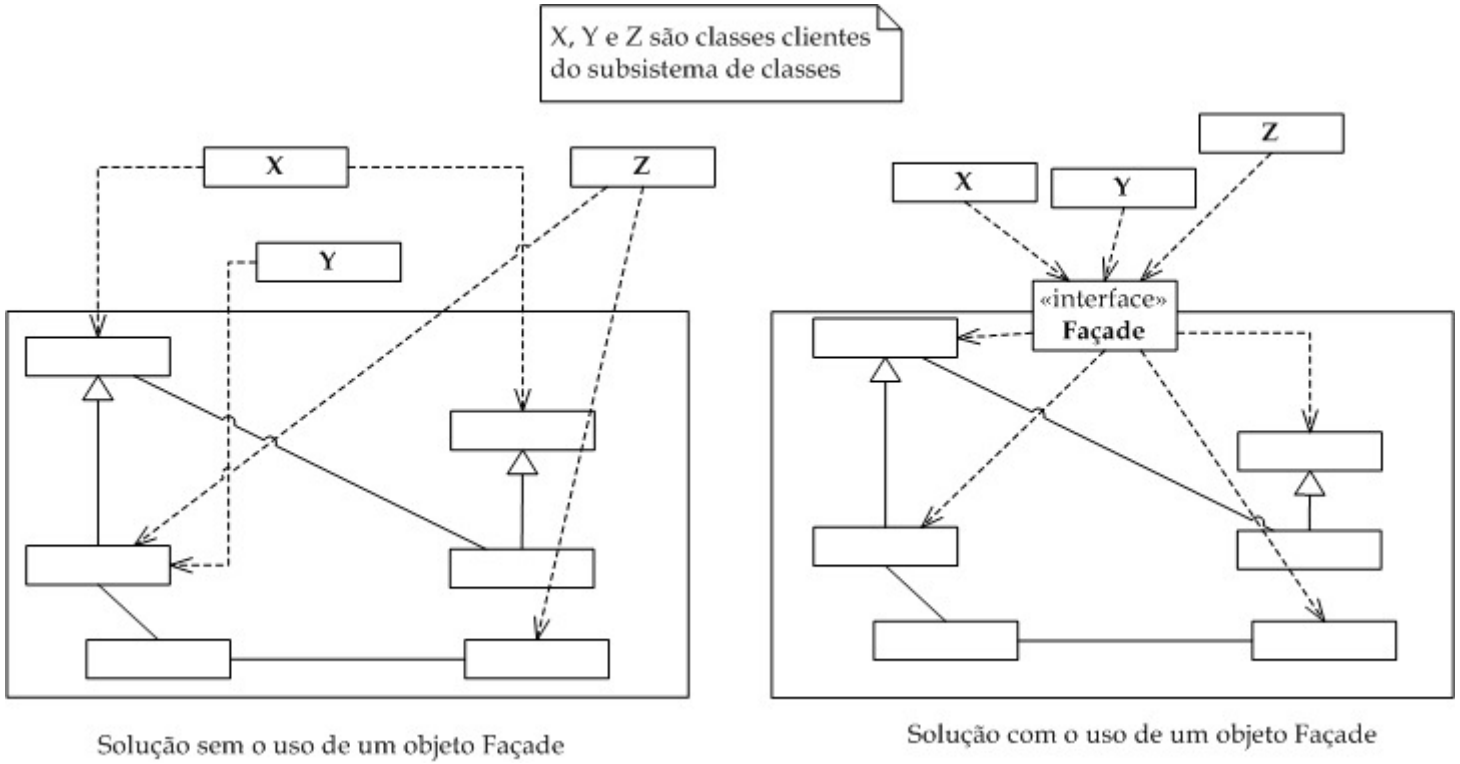
- Algumas observações sobre a implementação do padrão:
  - Mediador e Colega são classes abstratas ou interfaces.
  - A solução proposta pelo Mediator é definir um objeto, o *mediador*, para encapsular interações da seguinte forma: o resultado da interação de um subgrupo de objeto é passado a outro subgrupo pelo mediador.
  - Dessa forma, os subgrupos não precisam ter conhecimento da existência um do outro e podem variar independentemente.
  - *Objetos de controle* são exemplos de mediadores.

Façade

- Categoria: Padrões Estruturais
- Problema: estruturar um sistema em subsistemas ajuda a reduzir a complexidade. Um objetivo comum de todos os projetos é minimizar a comunicação e as dependências entre subsistemas. Como definir uma interface de alto nível que torne um subsistema mais fácil de ser utilizado?
- Intenção: Fornecer uma interface unificada (uma fachada) para um conjunto de interfaces em um subsistema. Façade define uma interface de nível mais alto que torna o subsistema mais fácil de ser usado.

Façade

- Estrutura

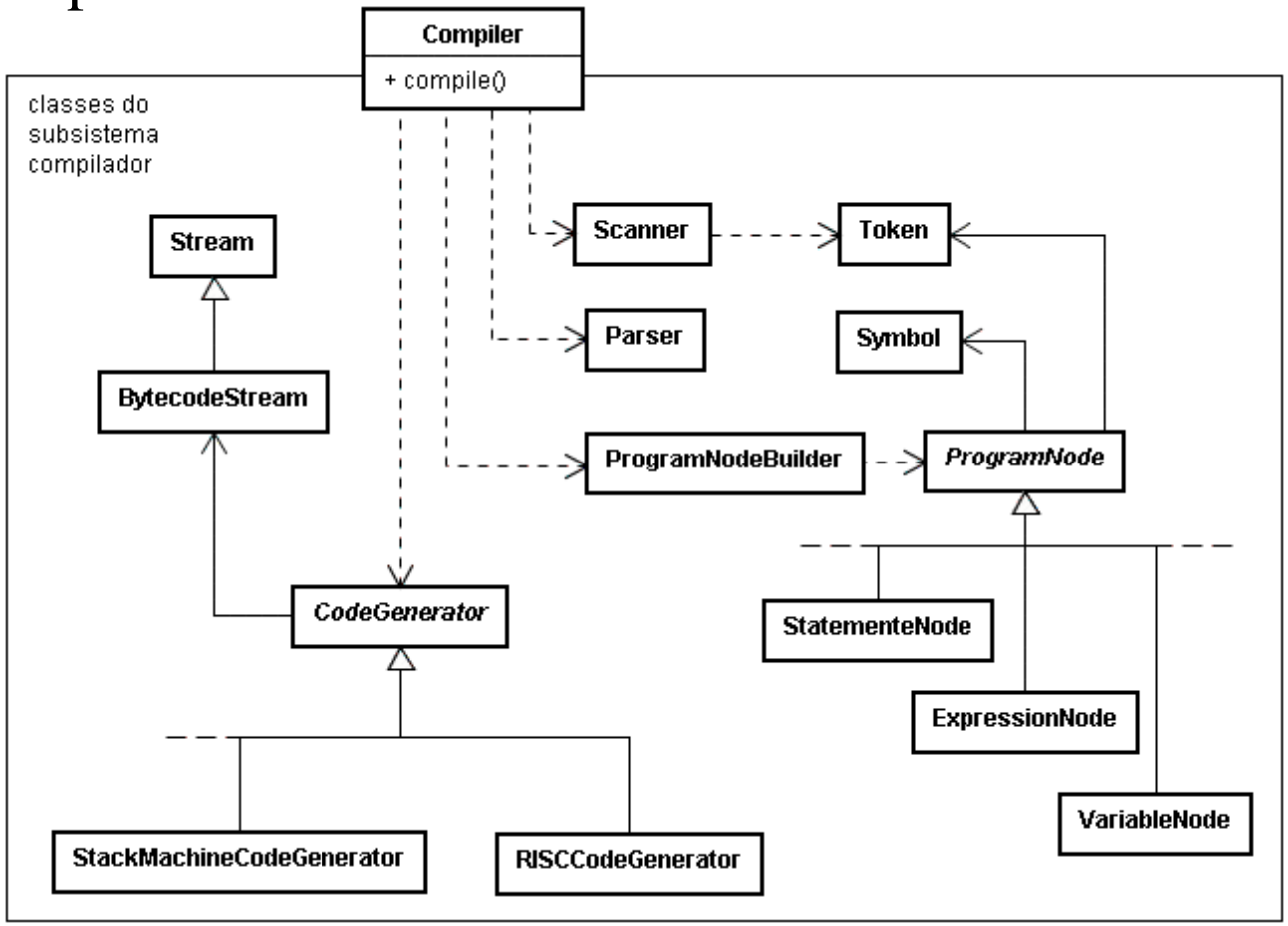


Solução sem o uso de um objeto Façade

Solução com o uso de um objeto Façade

Façade

- Exemplo



Façade

- Algumas observações sobre a implementação do padrão:
  - a vantagem da solução utilizando uma fachada é que o cliente somente conhece o necessário e suficiente em relação à complexidade do fornecedor; toda a complexidade adicional desse último fica “escondida” por trás da interface de comunicação;
  - a implementação do padrão Façade é relativamente simples; consite em definir uma ou mais classes que implementam a interface de comunicação necessária e suficiente; o subsistema fornecedor fica encapsulado por essa interface.