Instituto de computação

Padrões GoF

Leonardo Gresta Paulino Murta

leomurta@ic.uff.br

Agenda

- Introdução
- Padrões de Criação
- Padrões de Estrutura
- Padrões de comportamento

Introdução

- Os padrões GoF (Gamma et al., 1994) formam um catálogo de boas decisões de projeto
- Este catálogo é dividido em três tipos de padrões:
  - **Padrões de criação**: preocupam-se em como criar objetos
  - **Padrões de estrutura**: preocupam-se em como compor objetos
  - **Padrões de comportamento**: preocupam-se em como os objetos devem interagir

Introdução

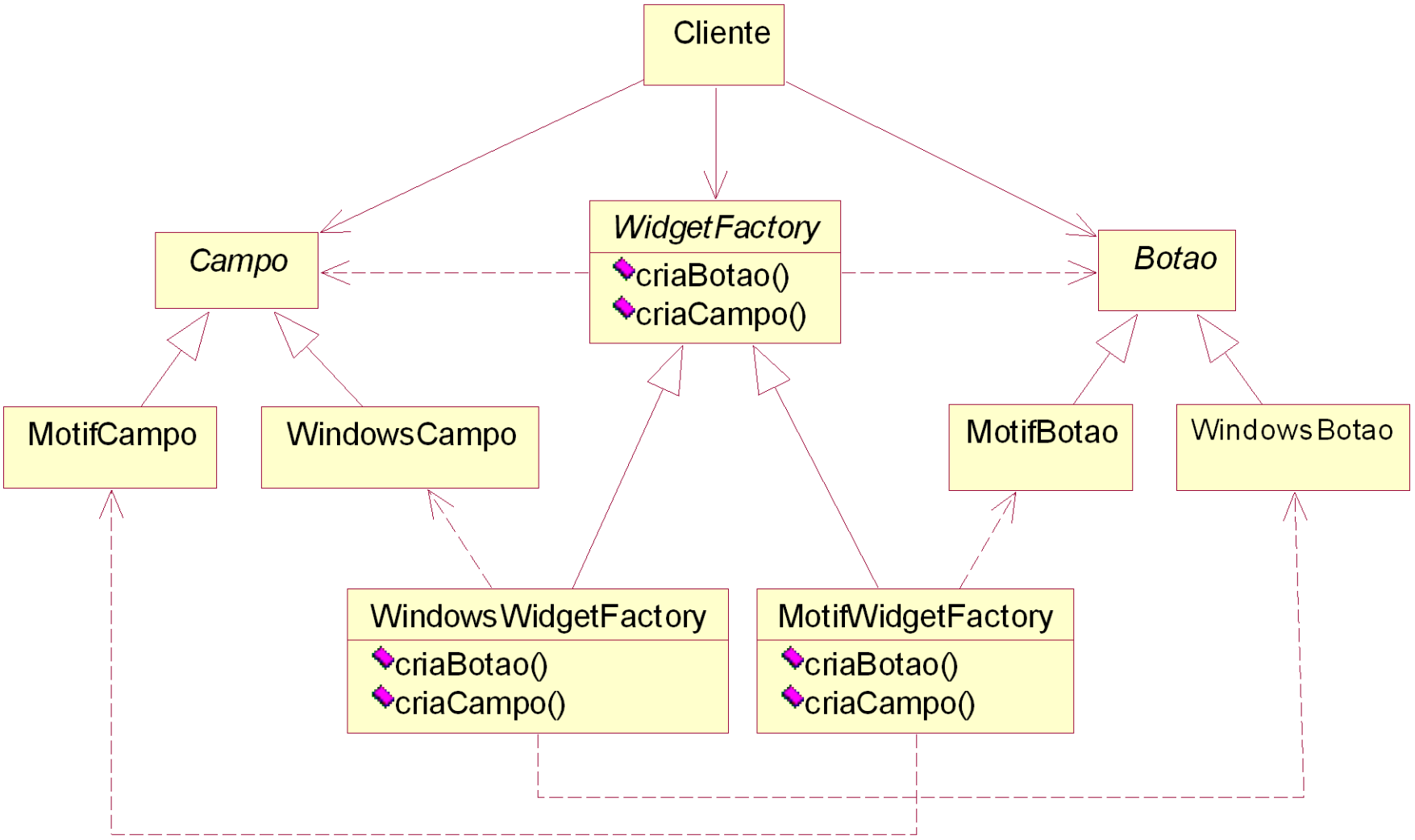
- Os padrões GoF refletem situações muito recorrentes em projeto OO, e podem ser vistos como o mínimo que todo projetista OO deveria saber
- Neste catálogo também está descrita a estrutura de documentação de um padrão e como os padrões se relacionam
- Existem vários outros catálogos de padrões
  - Esses catálogos relatam padrões em diferentes níveis de abstração: análise, arquitetura, projeto e codificação (idioma)

Padrões de Criação

Padrão *Abstract Factory*

- Objetivo:
  - Fornecer uma interface para criação de objetos relacionados sem especificar as suas classes concretas
- Motivação:
  - Suponha que se deseja fazer um sistema de janelas independente de SO (Motif, Windows, ...)
  - Seria necessário definir os *widgets* de forma abstrata e especializar para cada SO
  - Além disso, seria necessário definir uma fábrica genérica de *widgets* e especializar para os SOs

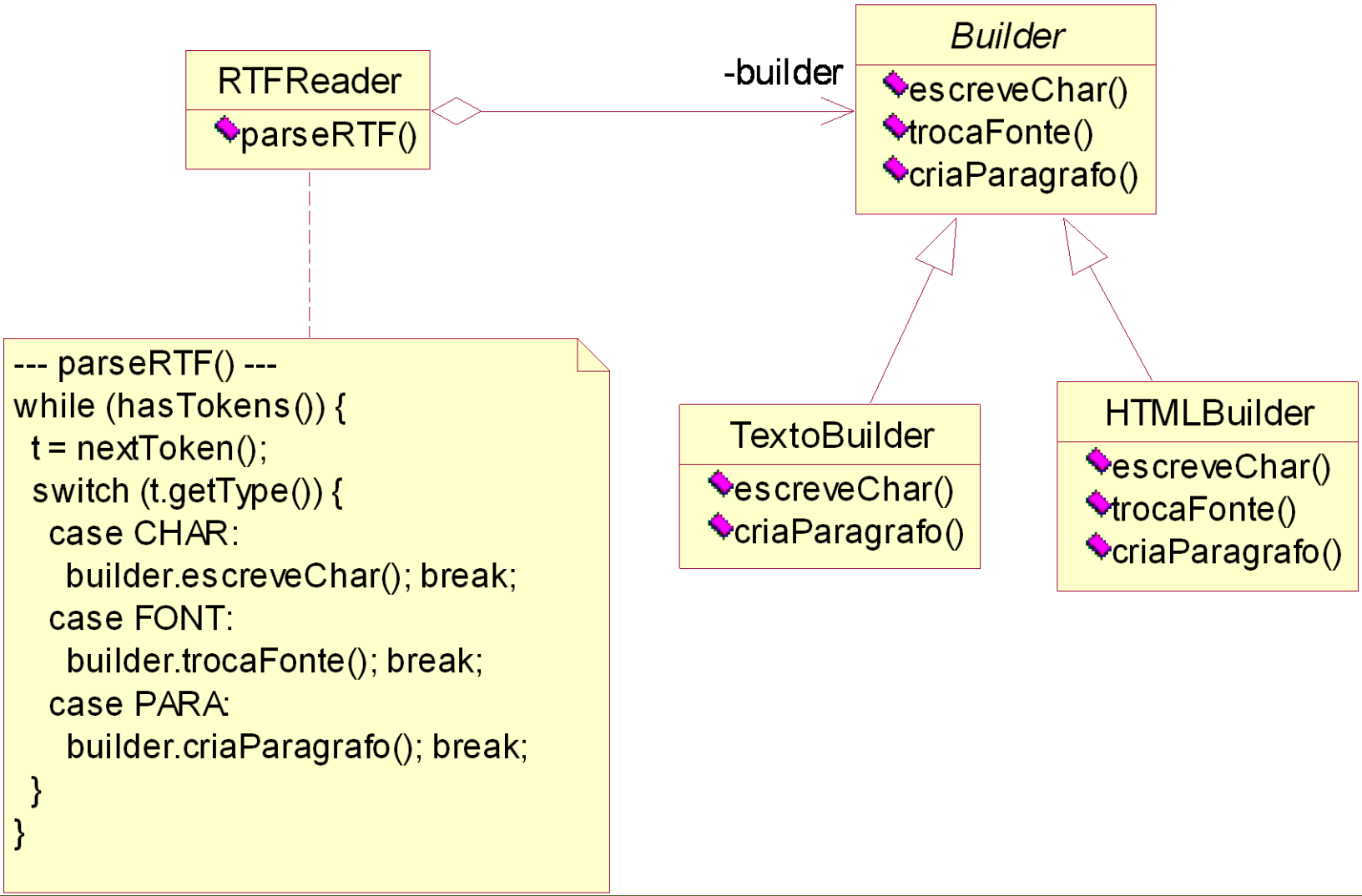
Padrão *Abstract Factory*



Padrão *Builder*

- Objetivo:
  - Separar a construção de objetos complexos da sua representação de forma que o mesmo processo de construção possa criar diferentes representações
- Motivação:
  - Suponha que se deseje converter documentos RTF para outros formatos (texto, HTML, ...)
  - Seria necessário verificar quais partes um documento pode ter e definir uma especialização para cada uma das partes

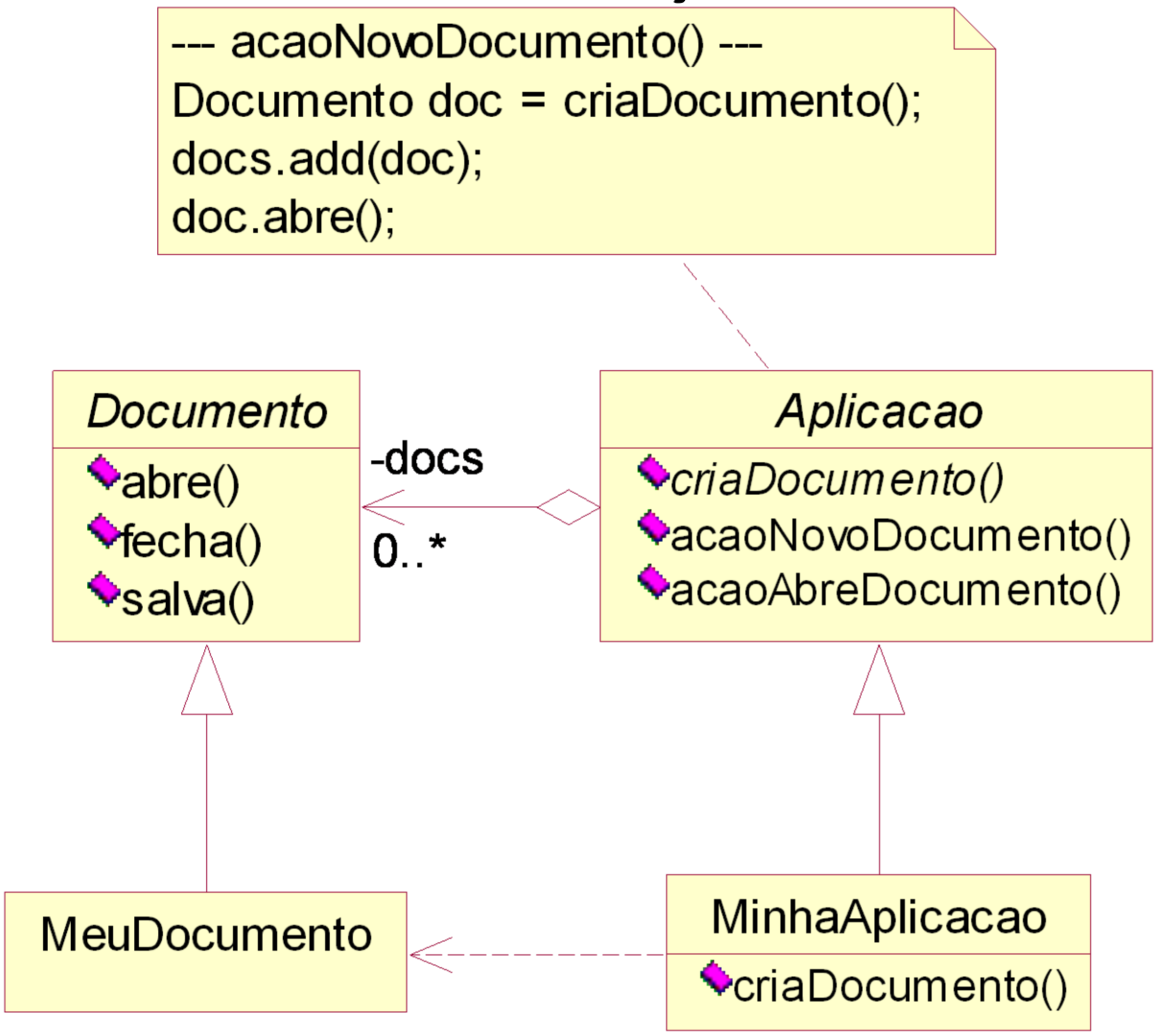
Padrão *Builder*



Padrão *Factory Method*

- Objetivo:
  - Definir uma interface de criação de objetos mas deixar as subclasses decidirem qual objeto criar
- Motivação:
  - Na construção de *frameworks*, não é possível, a priori, determinar qual elemento deve ser criado
  - A solução é permitir que o instanciador do *framework* faça a criação de uma instância específica

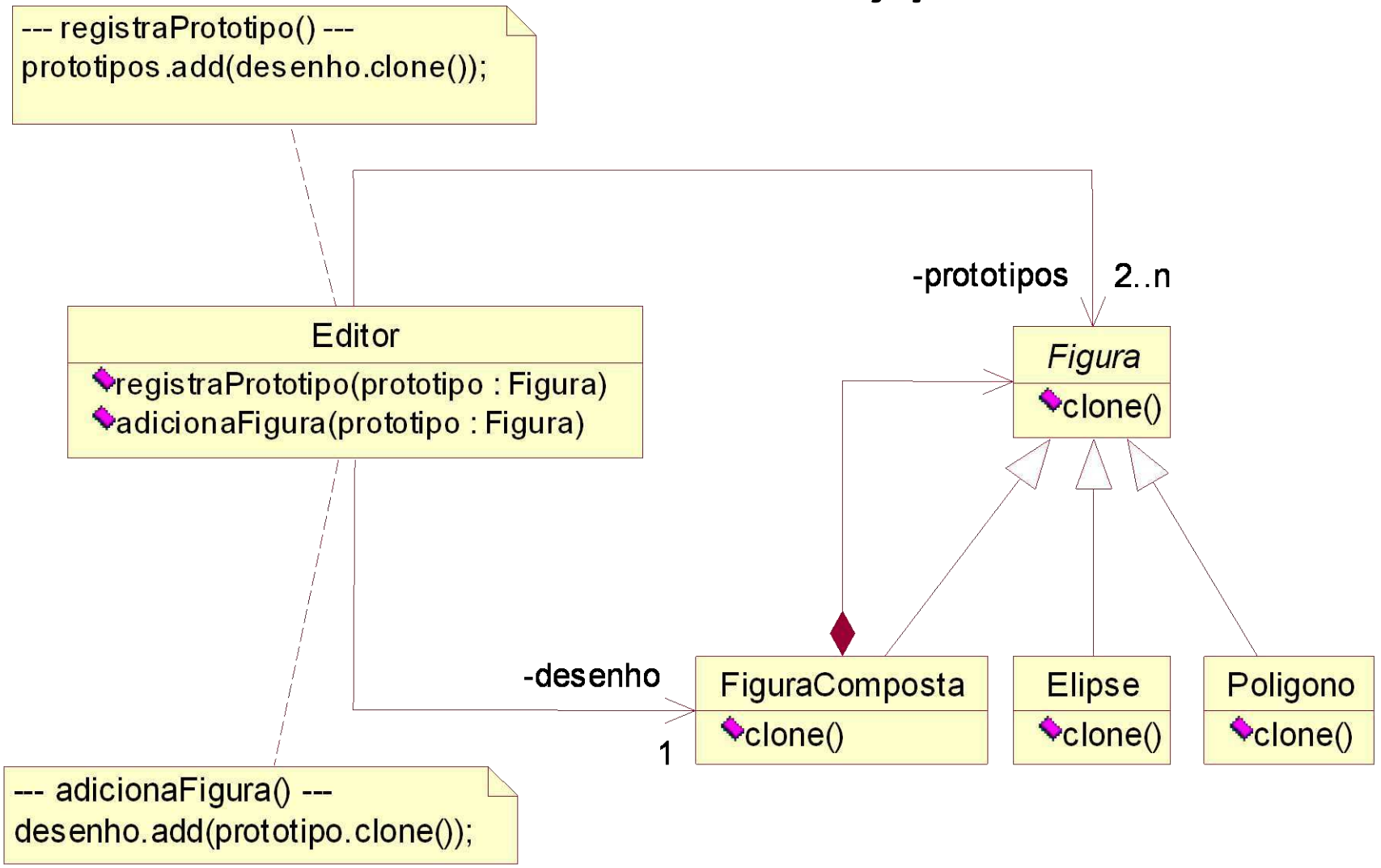
Padrão *Factory Method*



Padrão *Prototype*

- Objetivo:
  - Especificar tipos de objetos a serem criados usando protótipos e aplicando clonagem nesses protótipos
- Motivação:
  - Existem situações onde é desejável utilizar um objeto modelo complexo para a criação de outros objetos
  - Supondo o caso de uma ferramenta de desenho, pode ser útil criar estruturas complexas e fazer uso delas como base para outros desenhos mais complexos

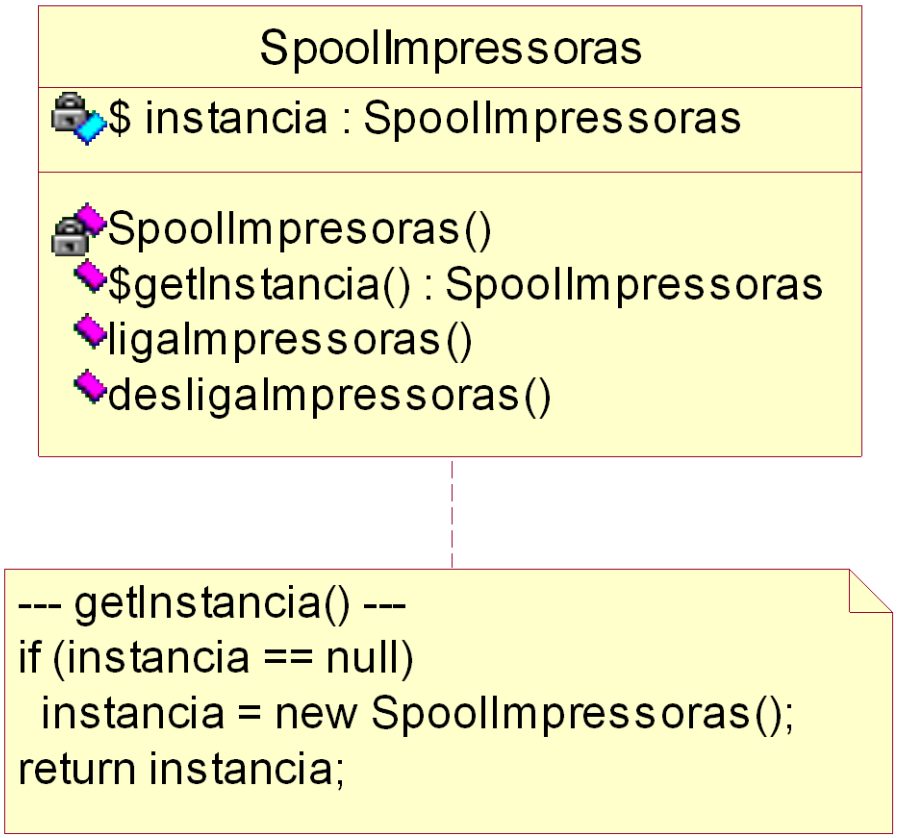
Padrão *Prototype*



Padrão *Singleton*

- Objetivo:
  - Assegurar que uma determinada classe tem somente uma instância e fornecer um ponto global de acesso a ela
- Motivação:
  - Em quase todo tipo de sistema existem classes com uma única instância
  - Em um sistema operacional, existe a necessidade de representar o *spool* de impressoras

Padrão *Singleton*

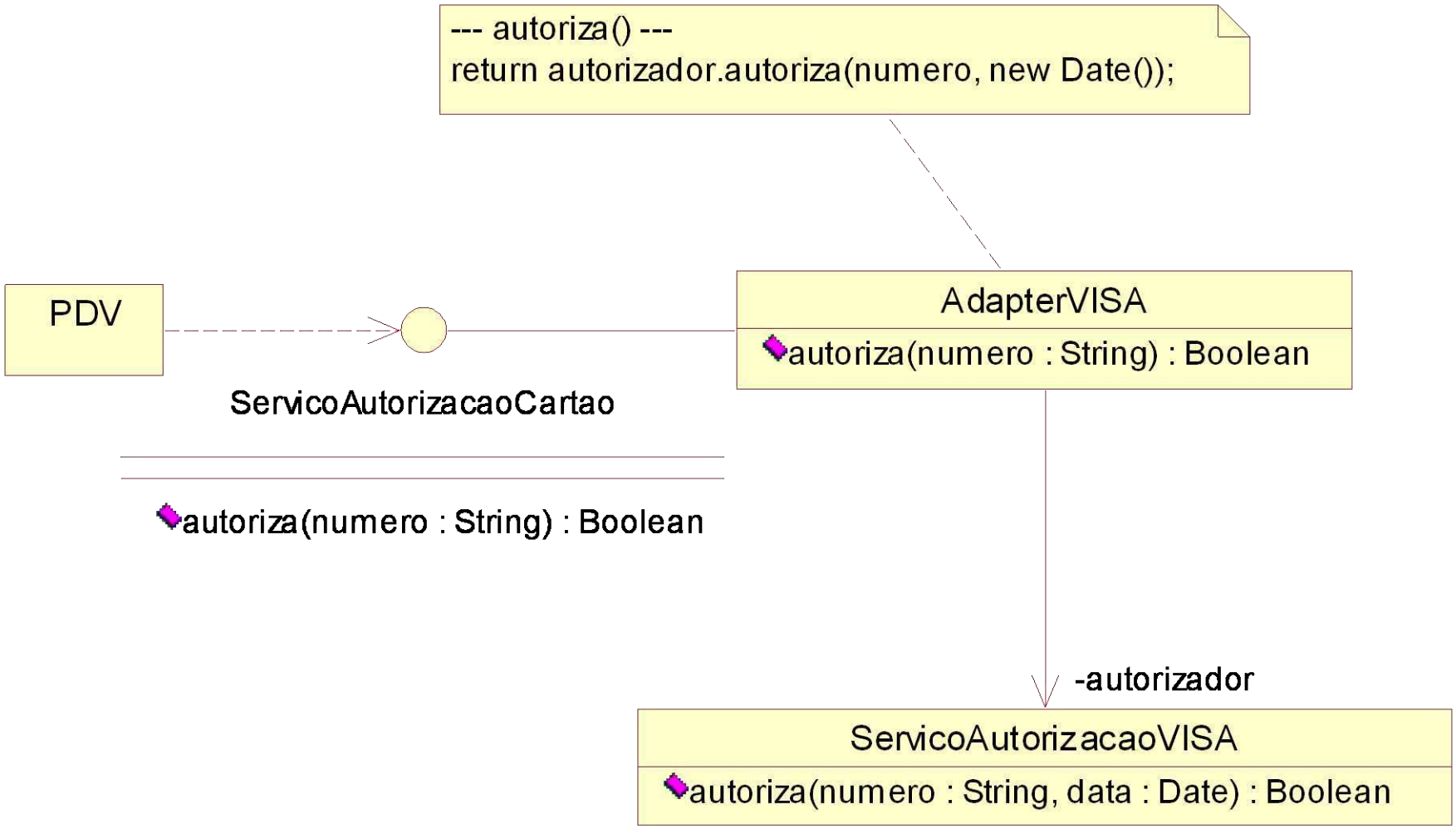


Padrões de Estrutura

Padrão *Adapter*

- Objetivo:
  - Converter a interface de uma classe em outra, para atender as expectativas do cliente
- Motivação:
  - Permitir que classes incompatíveis trabalhem em conjunto
  - É possível utilizar um *wrapper* para conectar o sistema de PDV com o serviço de autorização de cartão, mesmo que os métodos não sejam 100% compatíveis

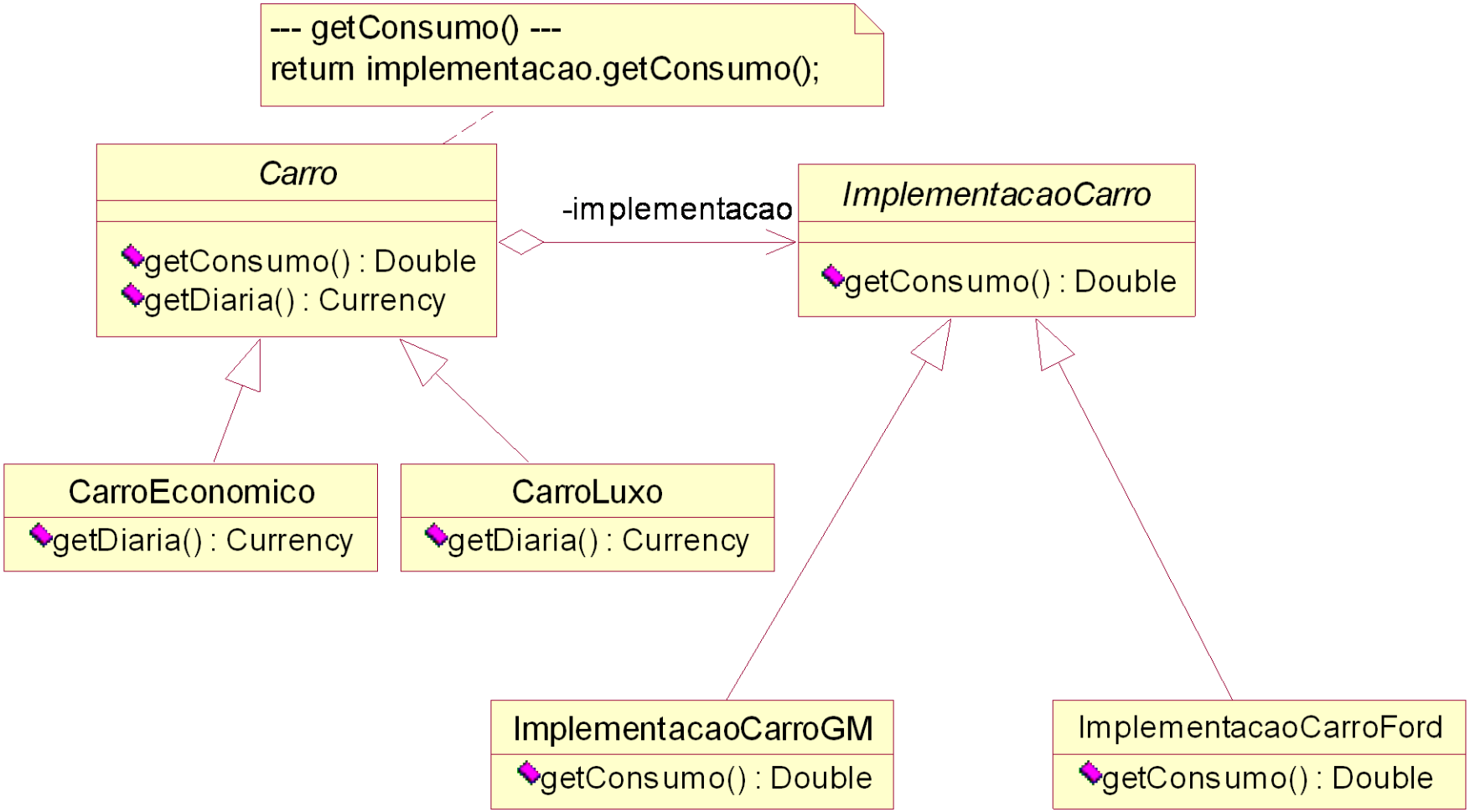
Padrão *Adapter*



Padrão *Bridge*

- Objetivo:
  - Desacoplar a abstração da sua implementação, de modo que ambos possam variar independentemente
- Motivação:
  - Supondo que um sistema de locadora de automóveis deseje representar categorias e modelos de carro
  - Será que CarroGM deve herdar de Carro? E CarroEconomico?
  - CarroGM é uma implementação de Carro

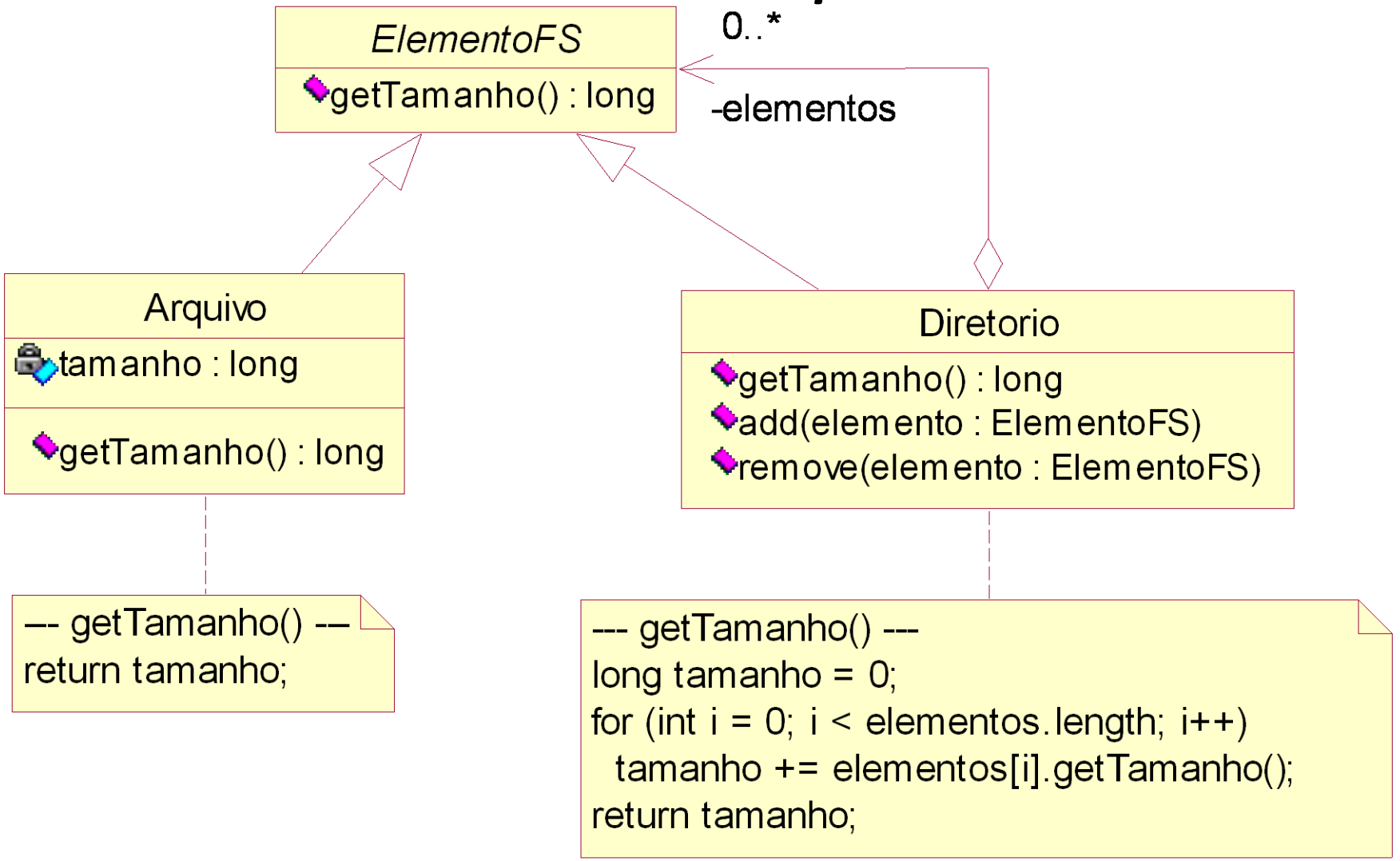
Padrão *Bridge*



Padrão *Composite*

- Objetivo:
  - Compor objetos utilizando uma estrutura de árvore para representar hierarquias de *todo-parte*
- Motivação:
  - Permitir que objetos do tipo *todo* ou do tipo *parte* sejam tratados da mesma maneira
  - No projeto de um sistema de arquivos, espera-se que tanto um arquivo quanto um diretório possam informar o seu tamanho

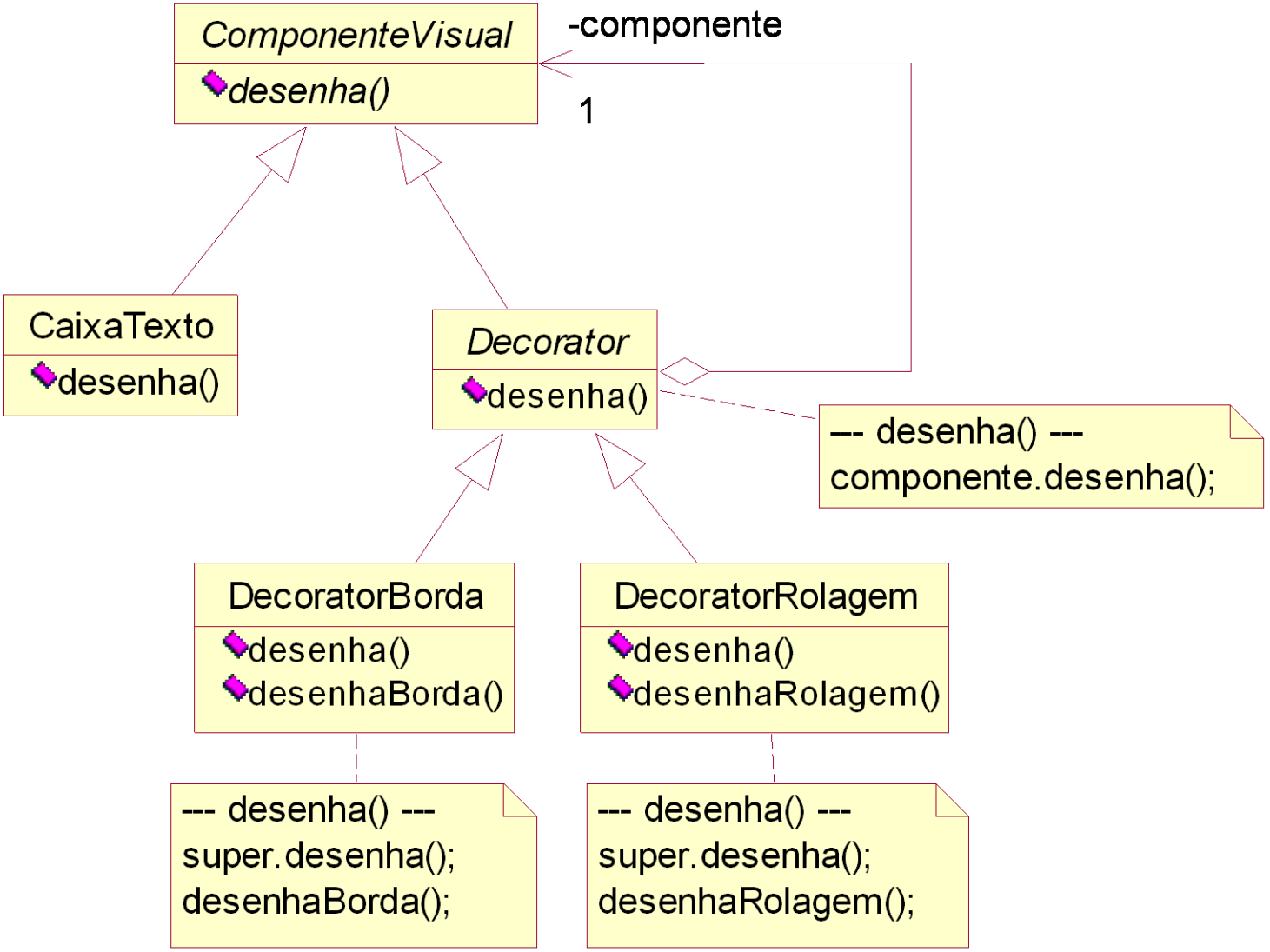
Padrão *Composite*



Padrão *Decorator*

- Objetivo:
  - Atribuir responsabilidades adicionais a um objeto de forma dinâmica
- Motivação:
  - Em algumas situações é desejado que um objeto tenha mais responsabilidades que os demais da sua classe
  - Supondo um sistema de janelas, é possível desejar que uma determinada caixa de texto tenha borda, e que outra tenha barra de rolagem

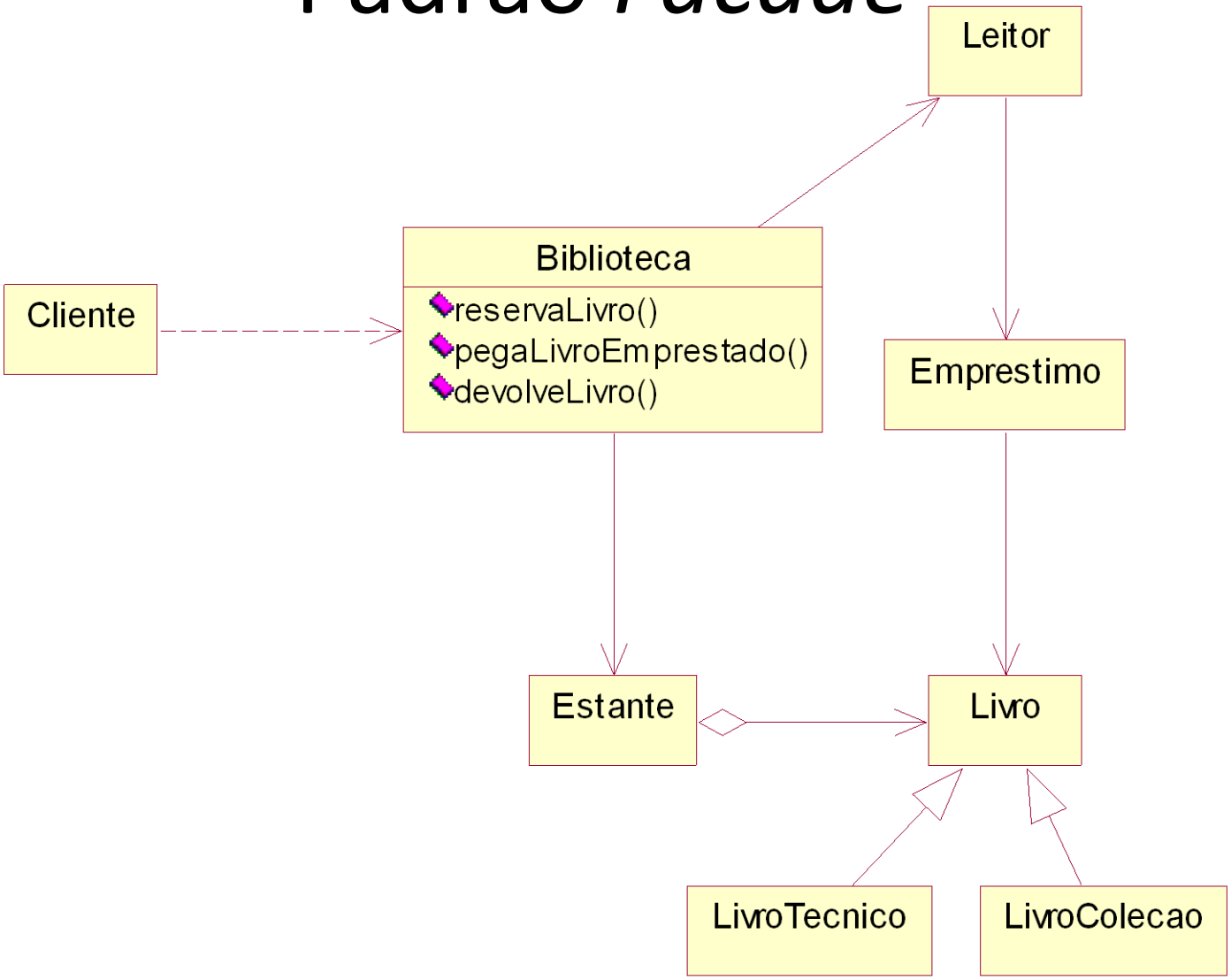
Padrão *Decorator*



Padrão *Facade*

- Objetivo:
  - Prover uma interface única para um conjunto de interfaces de um subsistema, facilitando o seu uso
- Motivação:
  - Existem situações onde um conjunto de classes deve se comportar como um componente
  - Para isso, o cliente deve falar com uma única interface
  - Suponha a interação entre um sistema cliente com um sistema de controle de biblioteca

Padrão *Facade*

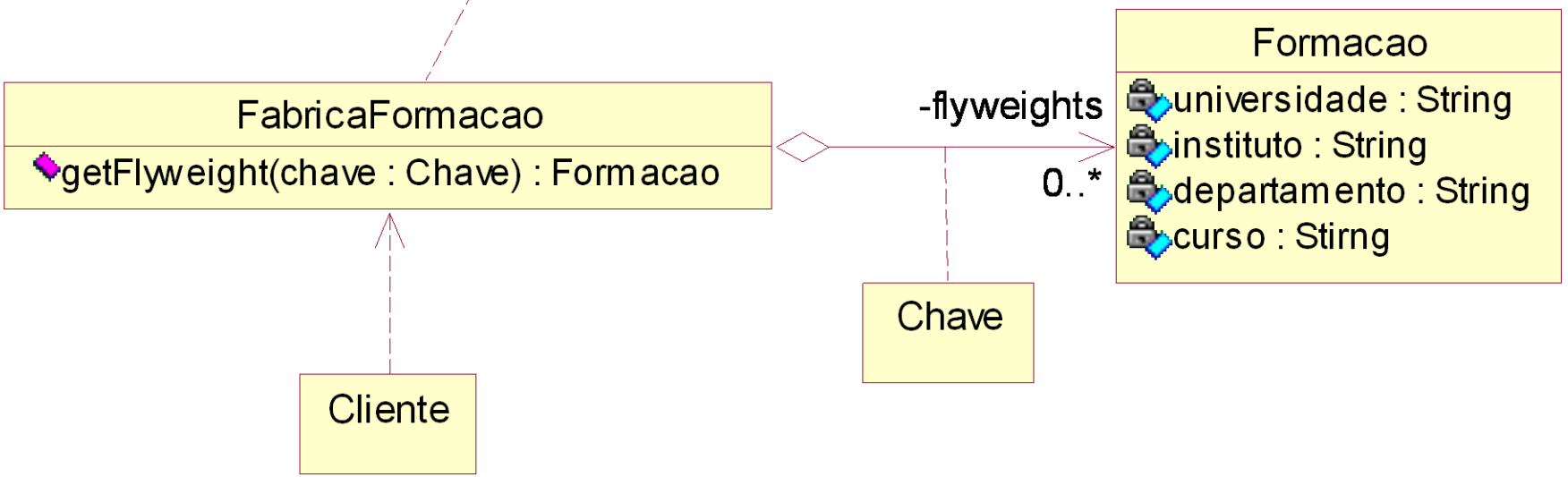


Padrão *Flyweight*

- Objetivo:
  - Utilizar compartilhamento para atender a um grande número de objetos
- Motivação:
  - Regularmente, objetos com conteúdos idênticos são criados em um sistema (para representar a mesma entidade)
  - Em um sistema de RH, o objeto que representa a formação dos funcionários pode estar sendo repetido

Padrão *Flyweight*

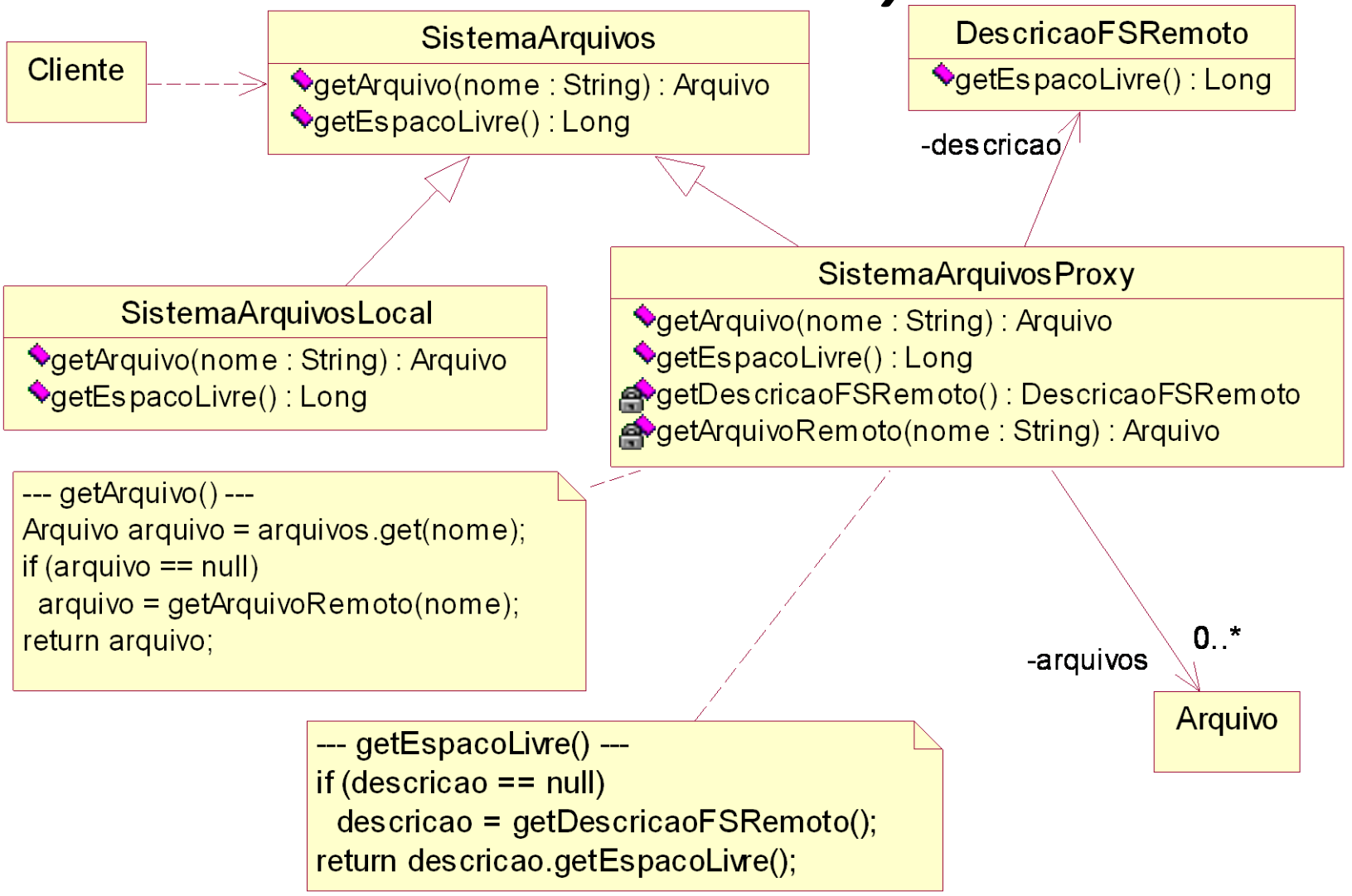
```
--- getFlyweight() ---
Formacao flyweight = (Formacao)flyweights.get(chave);
if (flyweight == null)
{
  flyweight = new Formacao();
  flyweights.put(chave, flyweight);
}
return flyweight;
```



Padrão *Proxy*

- Objetivo:
  - Fornece um substituto a um objeto (procurador)
- Motivação:
  - Em situações onde é custoso o acesso direto a um objeto (rede, disco, etc.), pode-se utilizar um representante local
  - Em um sistema operacional, é necessário fornecer acesso a sistemas de arquivos remotos (ex.: NFS)

Padrão *Proxy*

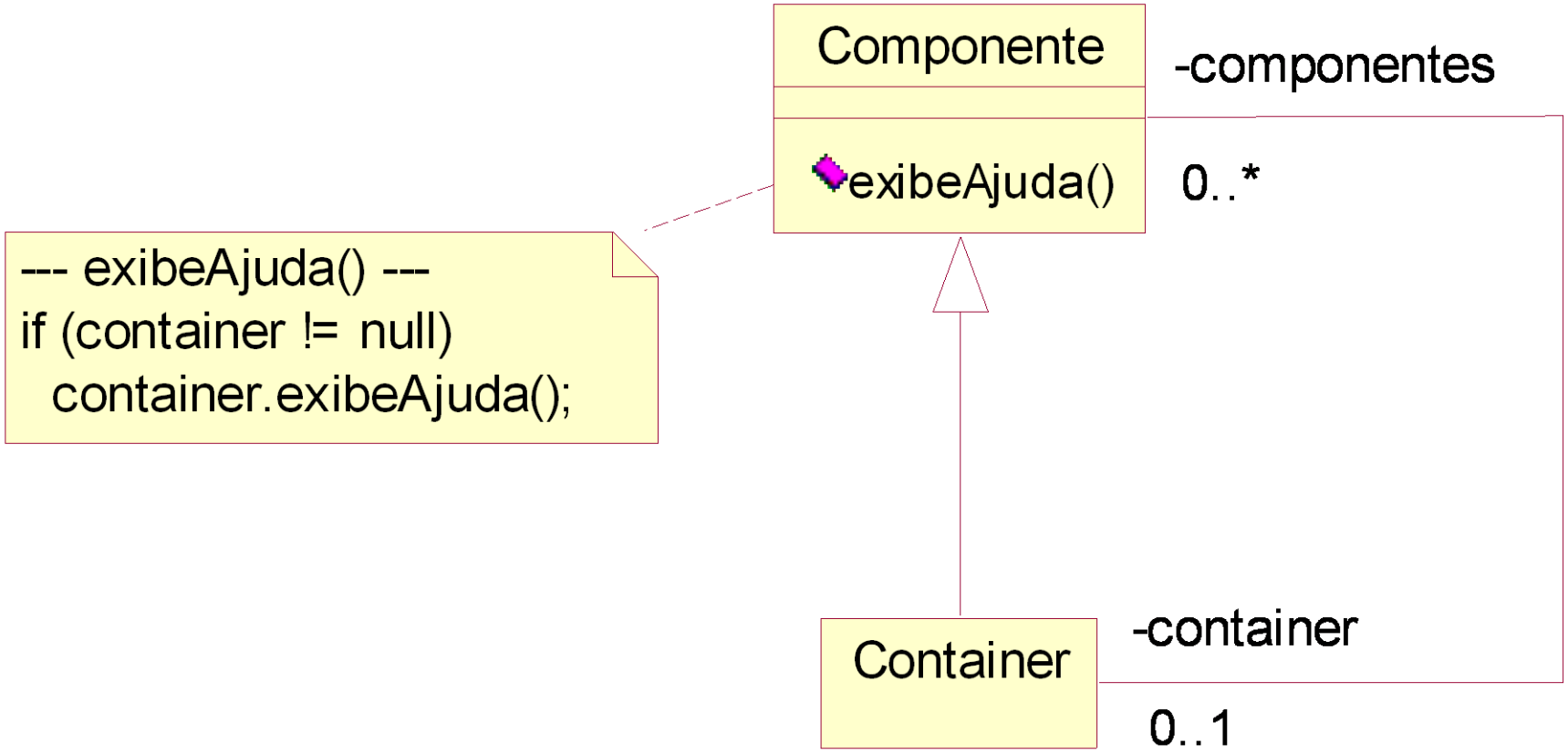


Padrões de Comportamento

Padrão *Chain of Responsability*

- Objetivo:
  - Desacoplar o objeto que envia uma requisição de um receptor único
- Motivação:
  - Permitir que mais de um objeto possa tratar uma requisição
  - Em um sistema com ajuda (*help*) sensível ao contexto, os componentes que não têm texto associado podem repassar a responsabilidade para os seus *containers*

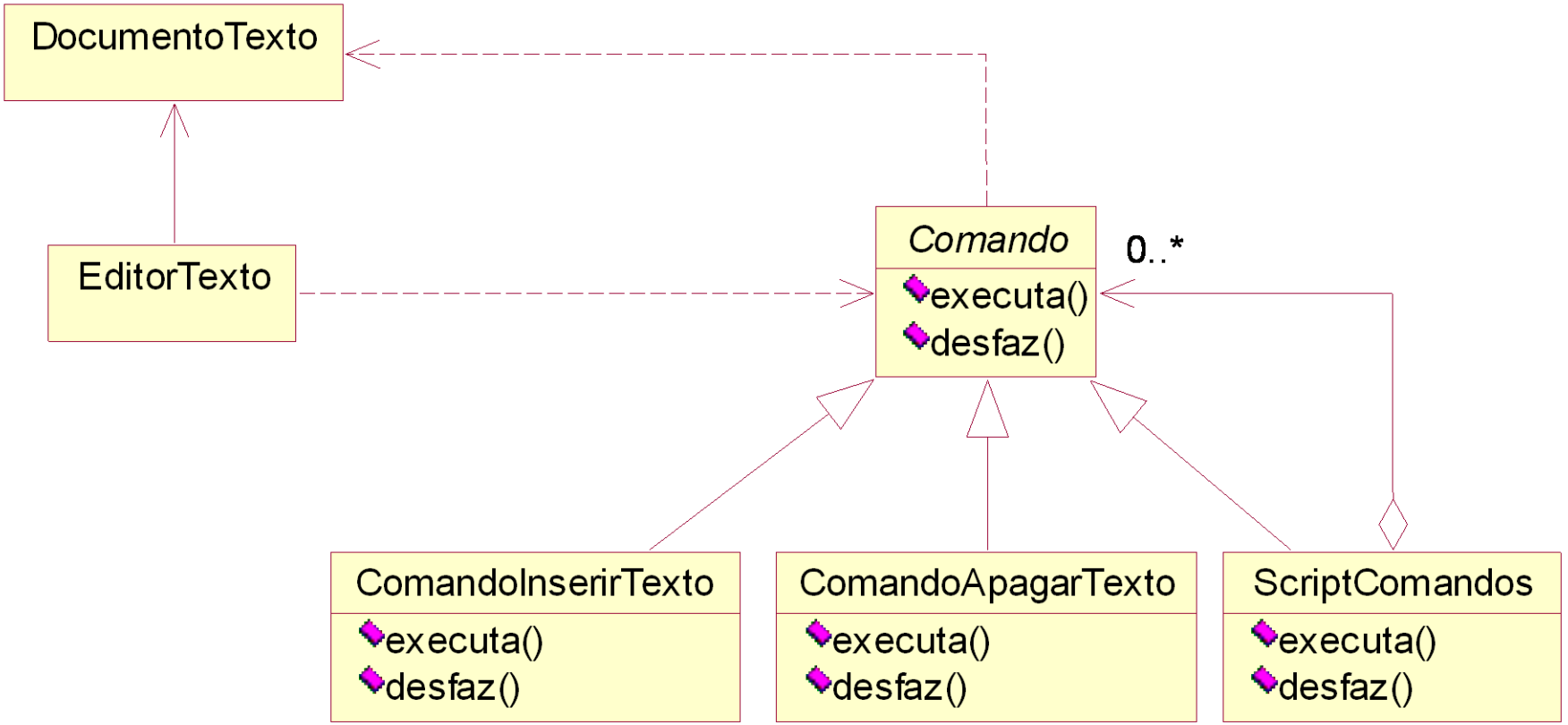
Padrão *Chain of Responsability*



Padrão *Command*

- Objetivo:
  - Encapsular comandos em objetos para permitir utilização em lote ou *undo* dos mesmos
- Motivação:
  - Em um sistema de edição de texto é desejável que seja possível desfazer uma ou mais ações efetuadas
  - Também é desejável que o editor possa funcionar de forma não interativa, através de um roteiro (*script*) de comandos

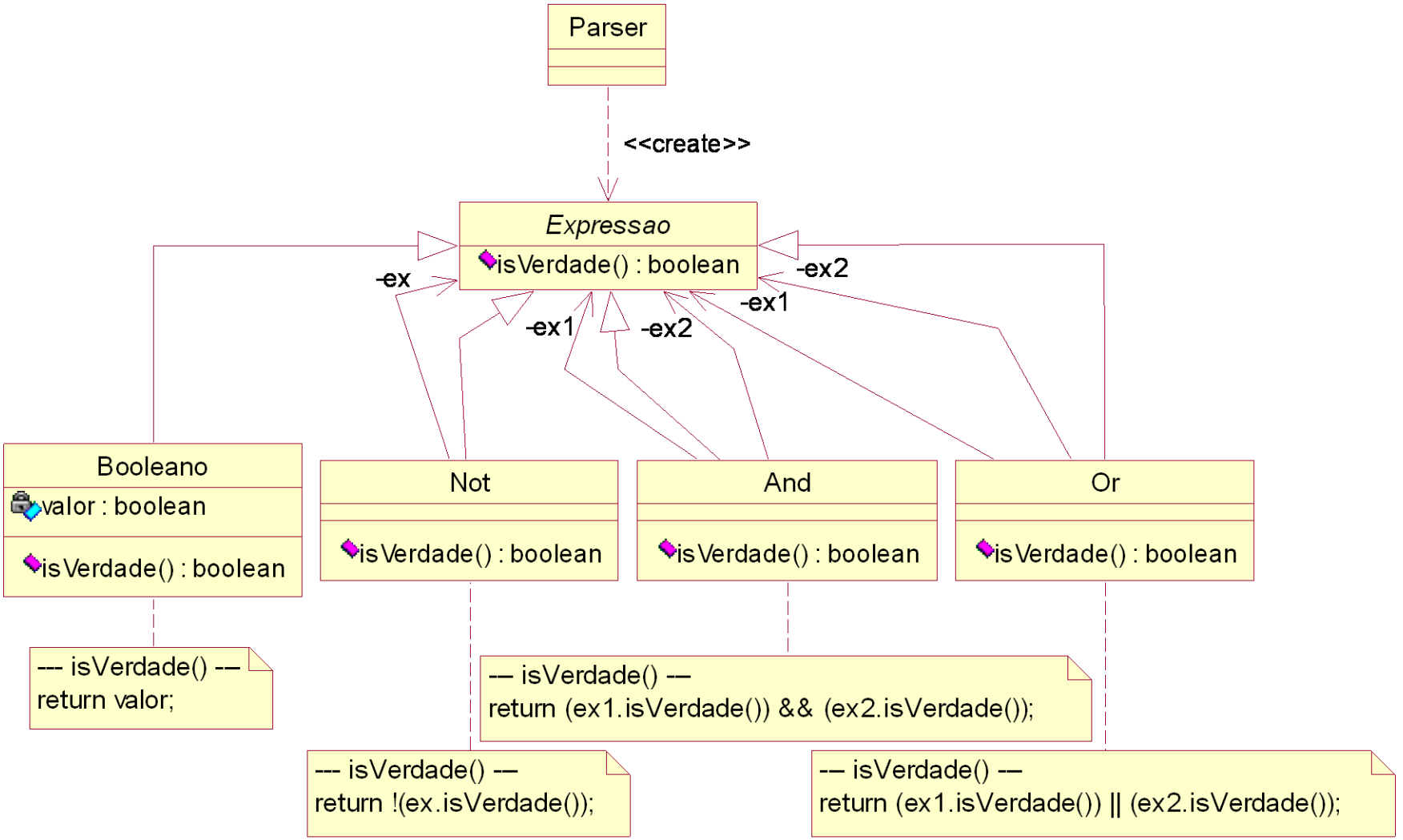
Padrão *Command*



Padrão *Interpreter*

- Objetivo:
  - Definir uma representação em objetos de uma determinada linguagem
- Motivação:
  - Em um sistema que lida com expressões lógicas, é interessante a criação de uma linguagem que facilite a sua utilização
  - Por exemplo, uma expressão (Ex) pode ser representada por um Booleano, Not(Ex), And(Ex, Ex), Or(Ex, Ex)

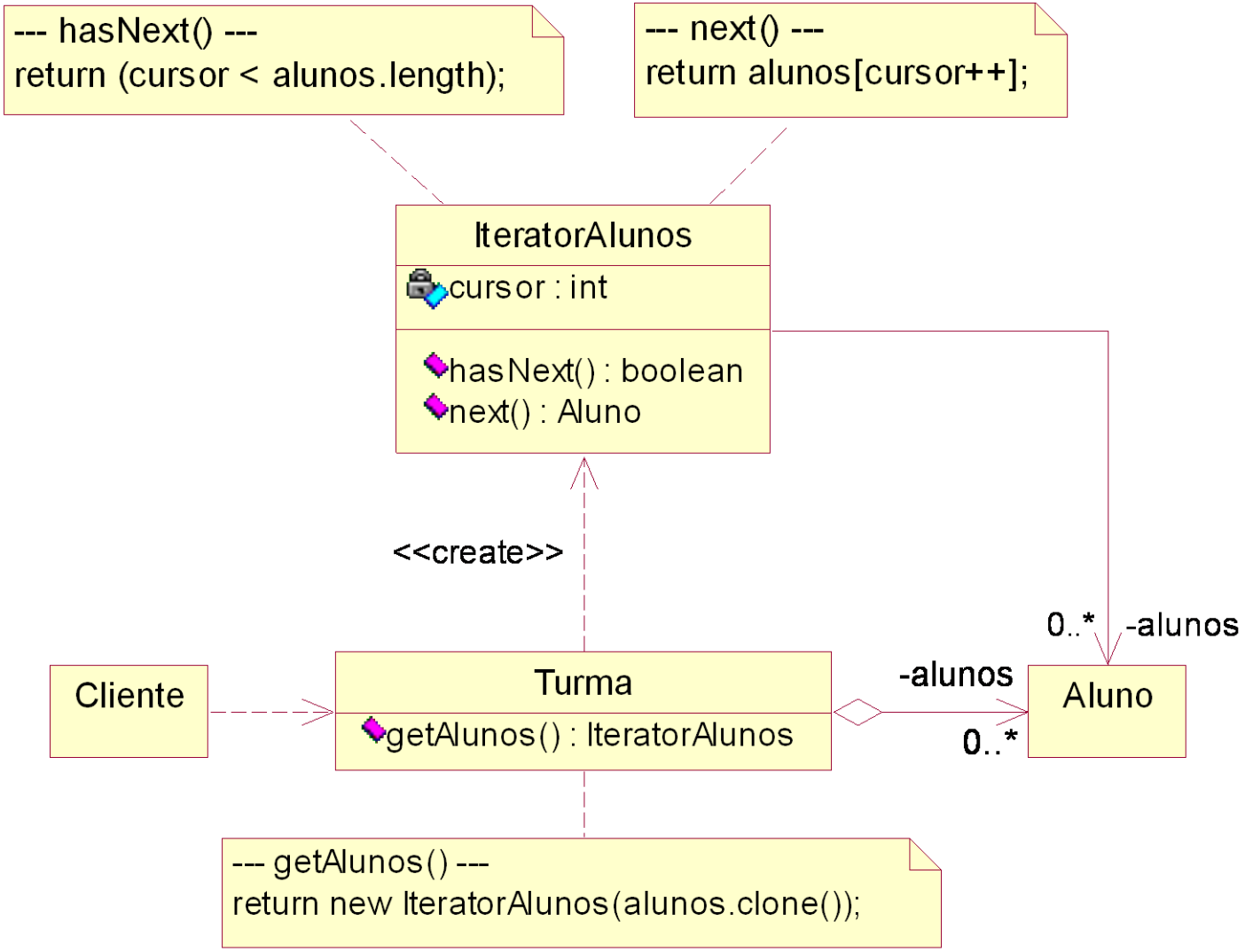
Padrão *Interpreter*



Padrão *Iterator*

- Objetivo:
  - Fornece um mecanismo de acesso a elementos de uma agregação sem quebrar o encapsulamento
- Motivação:
  - Se um objeto de Turma fornecer a sua lista de alunos interna, poderão ser removidos ou inseridos alunos sem o conhecimento da Turma
  - Esse tipo de problema ocorre com freqüência, e é visto como uma quebra de encapsulamento

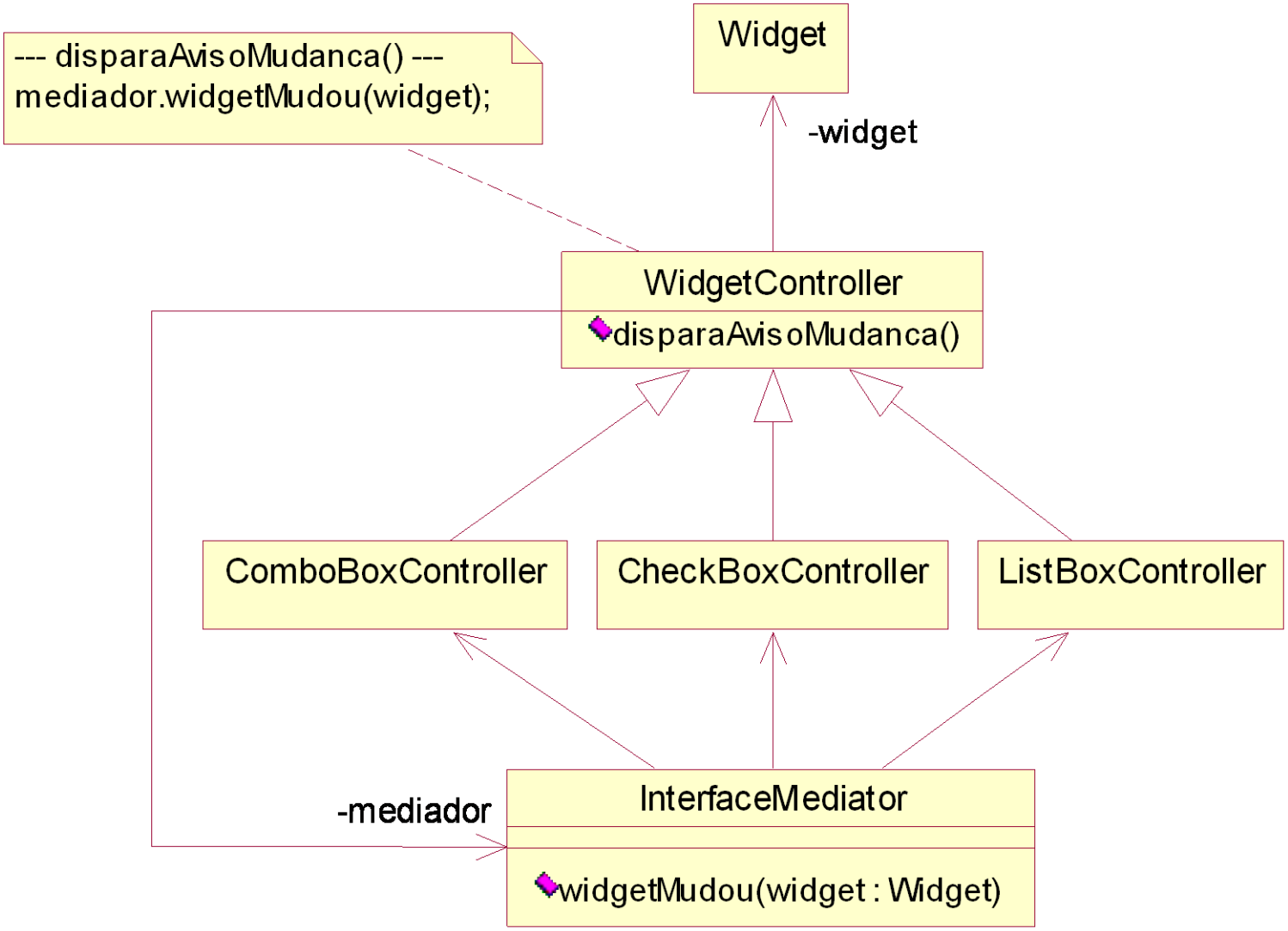
Padrão *Iterator*



Padrão *Mediator*

- Objetivo:
  - Definir um objeto que encapsula como vários outros objetos interagem
- Motivação:
  - Promover acoplamento fraco não permitindo que os objetos se referenciem
  - Em um sistema de informação, um Mediator pode ser utilizado para manter a consistência da interface
  - Ex.: habilitar ou desabilitar opções em função da seleção de outras opções

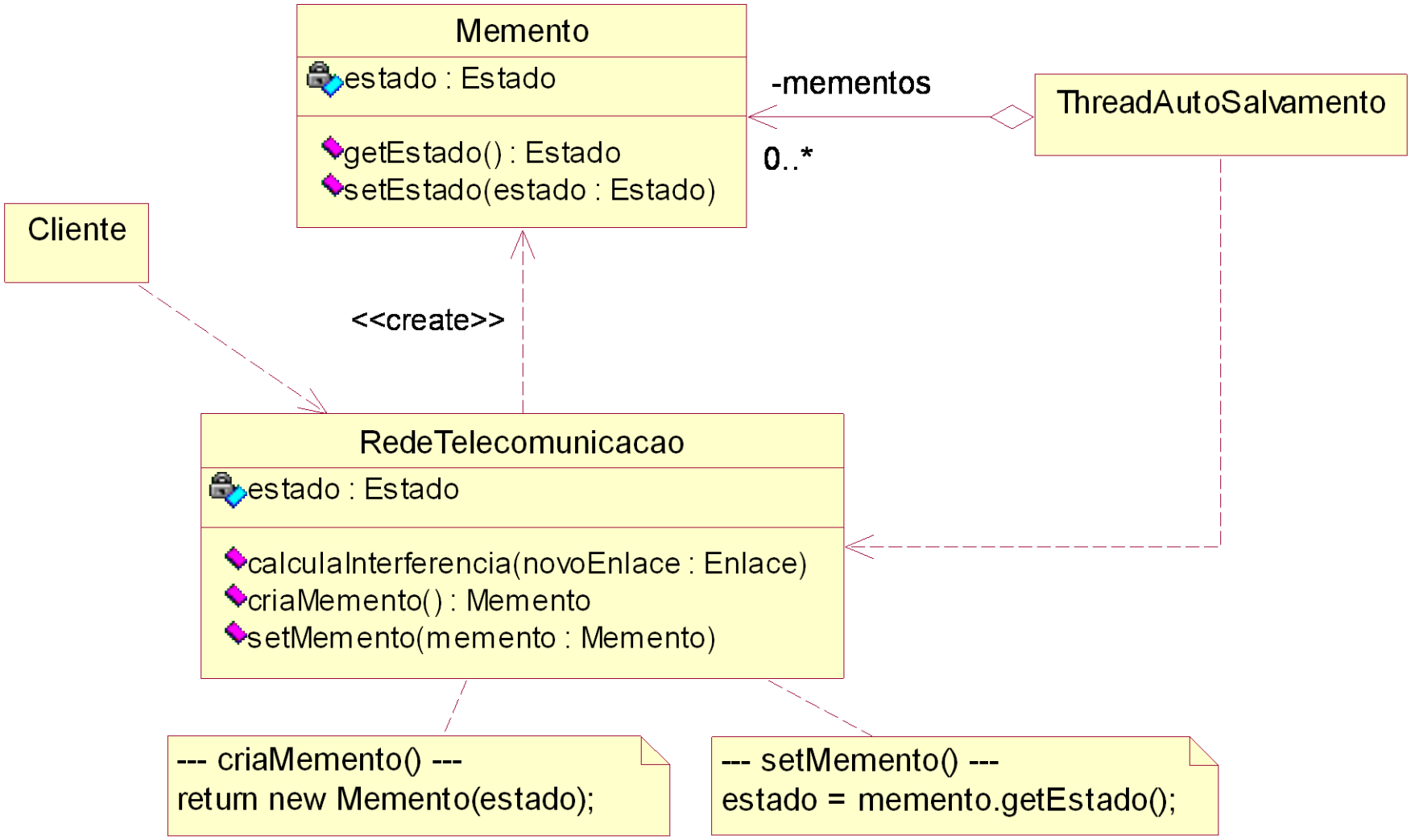
Padrão *Mediator*



Padrão *Memento*

- Objetivo:
  - Capturar e externalizar o estado interno de um objeto sem quebrar o encapsulamento
- Motivação:
  - Permitir a implementação de mecanismos de *checkpoint* em um sistema para possibilitar *undo*
  - Útil em situações que o padrão *Command* não funciona
  - Em um processamento numérico longo, podem ser salvos estados intermediários

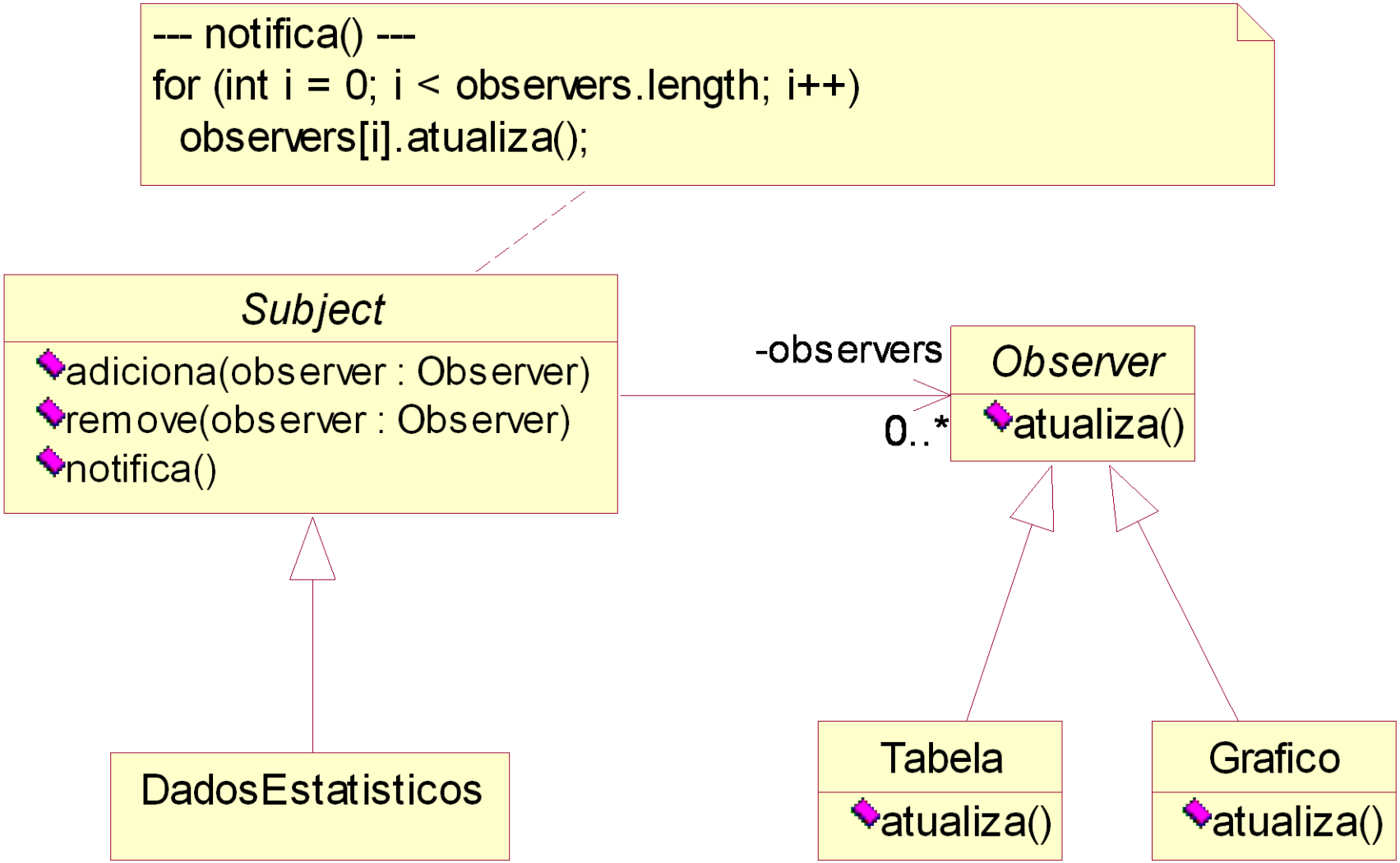
Padrão *Memento*



Padrão *Observer*

- Objetivo:
  - Definir uma dependência de um para muitos com um mecanismo de notificação de eventos
- Motivação:
  - A mesma informação (modelo) pode ser exibida em diferentes formatos (visão) em paralelo
  - Suponhamos que dados estatísticos devem ser exibidos, ao mesmo tempo, em forma de tabela e gráfico
  - Esses dados podem ser alterados durante a exibição

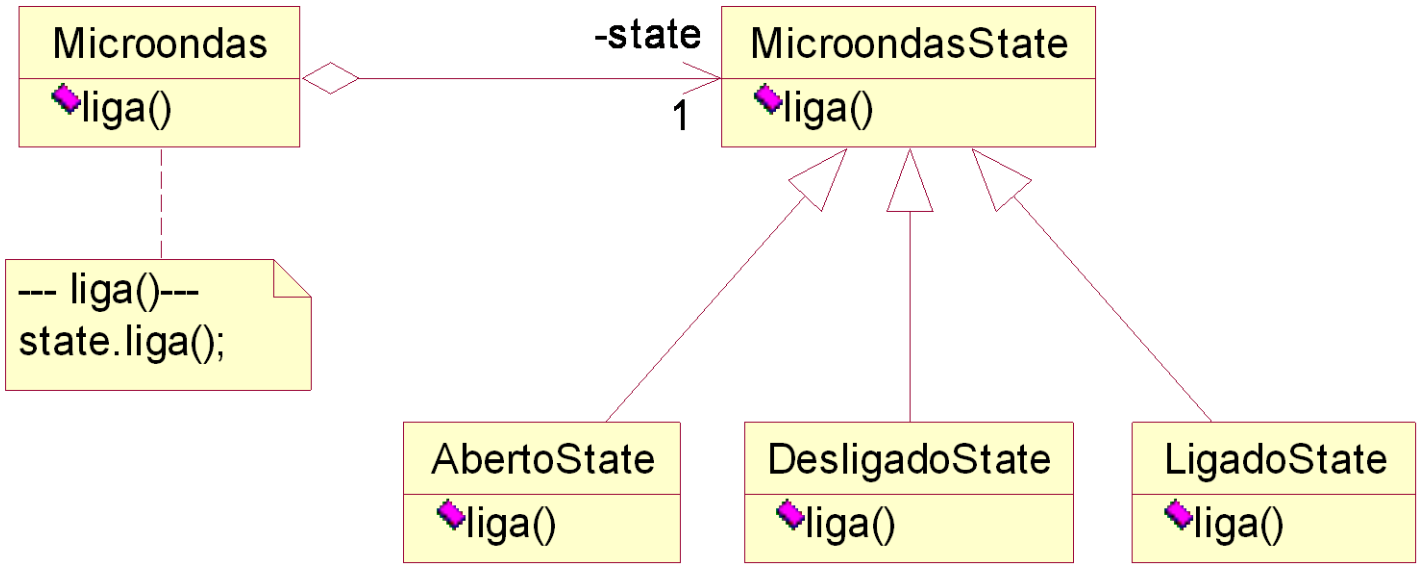
Padrão *Observer*



Padrão *State*

- Objetivo:
  - Permitir que um objeto modifique o seu comportamento em função do seu estado interno
- Motivação:
  - Gerar o efeito de uma troca de tipo de um objeto em tempo de execução
  - Um forno de microondas deve se comportar de formas diferentes em função do seu estado quando o botão de “ligar” é pressionado

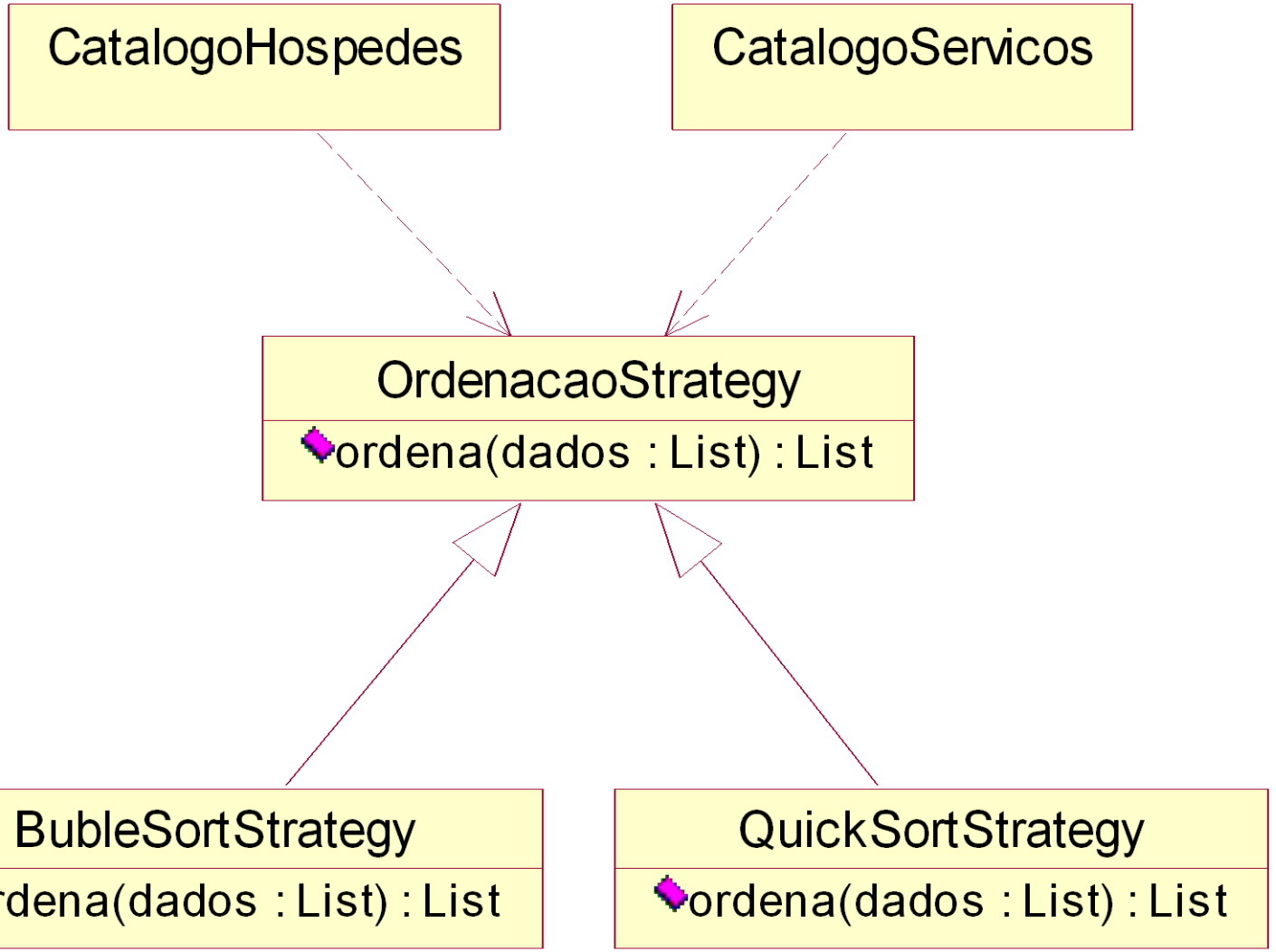
Padrão *State*



Padrão *Strategy*

- Objetivo:
  - Definir uma família de algoritmos de forma encapsulada
- Motivação:
  - Alguns algoritmos se repetem com freqüência, por isso devem ser isolados para facilitar a manutenção
  - Tanto os objetos da classe CatalogoHospedes quanto os objetos da classe CatalogoServicos de um sistema de controle de hotéis precisam de algoritmos de ordenação

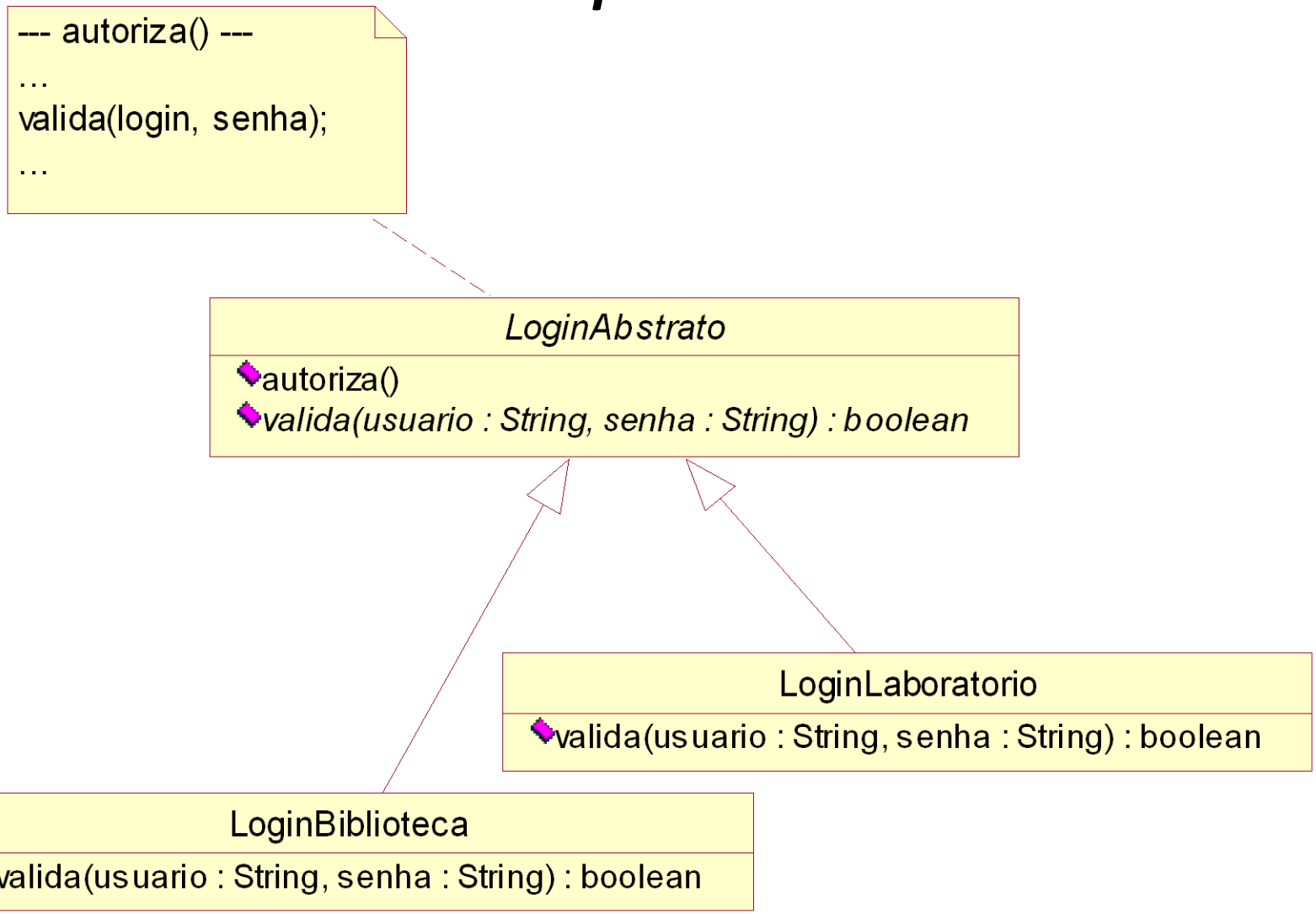
Padrão *Strategy*



Padrão *Template Method*

- Objetivo:
  - Definir o esqueleto de um algoritmo e delegar alguns dos seus passos para as subclasses
- Motivação:
  - Na construção de *frameworks* é necessário especificar o comportamento global e definir algumas lacunas para serem preenchidas no momento da instanciação
  - Em um *framework* de validação de usuário, a forma de validar pode ser SGBD, LDAP, arquivo, etc.

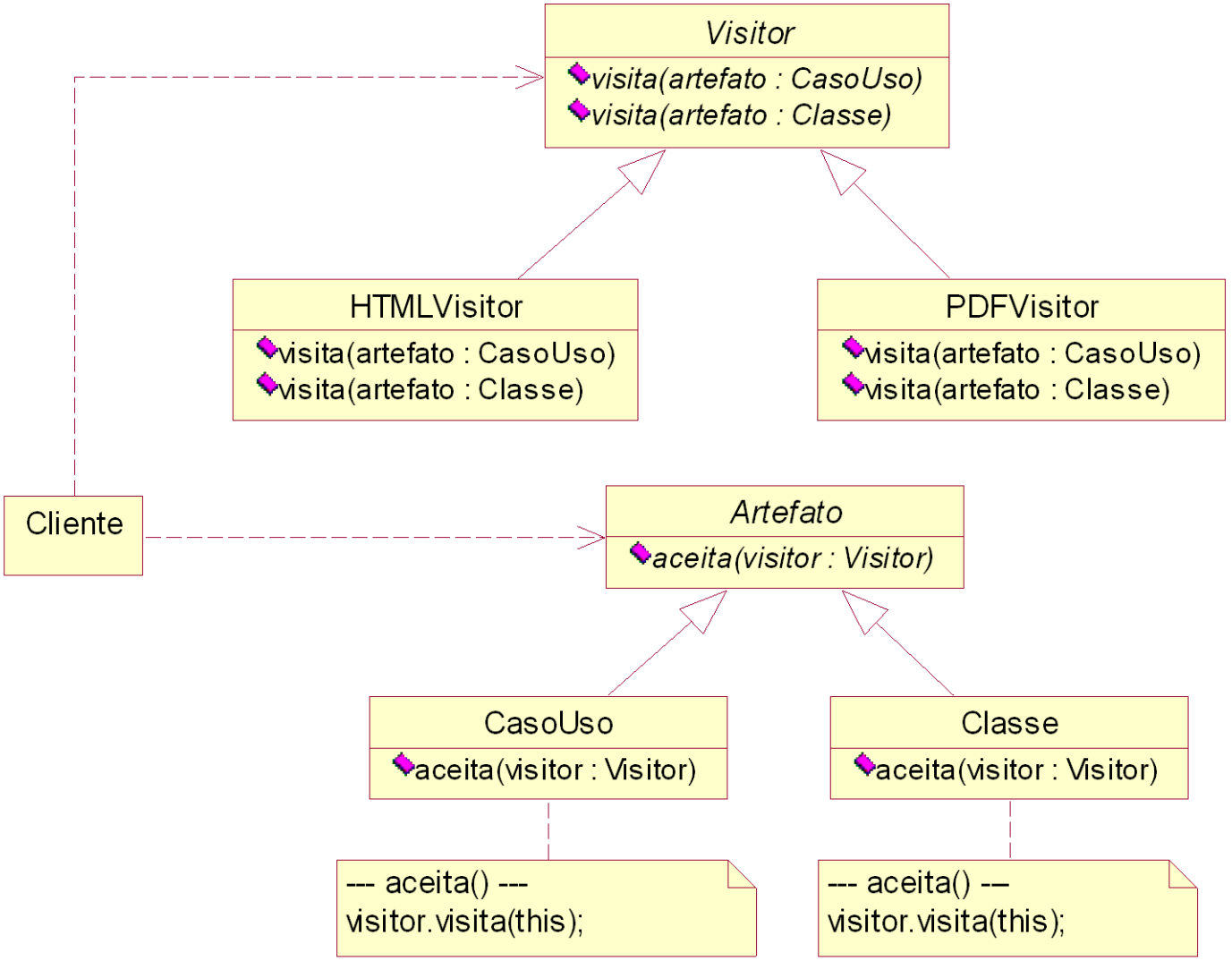
Padrão *Template Method*



Padrão *Visitor*

- Objetivo:
  - Aumentar a coesão das classes isolando em outras classes tarefas pouco coesas
- Motivação:
  - Em uma ferramenta Case deve ser possível gerar as informações de casos de uso e classes em HTML, PDF, RTF, TeX, etc.
  - Entretanto, não é desejável que a classe que representa um caso de uso tenha métodos para cada tipo de geração

Padrão *Visitor*



Bibliografia

- Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.; “Design Pattern – Elements of Reusable Object-Oriented Software”; Addison-Wesley; 1994

Instituto de computação

Padrões GoF

Leonardo Gresta Paulino Murta

leomurta@ic.uff.br