Especialização em Projeto e Desenvolvimento de Sistemas

Padrões de Projeto

Este material não substitui a leitura da bibliografia básica
Exemplos de código são de cunho didático

Prof. MSc.Vagner Figuerêdo de Santana


vsantana@ic.unicamp.br


www.plasticdesign.eti.br


@santanavagner



santanavagner



santana.vagner@gmail.com

Avaliação

- Exercícios
- Lista de exercícios durante a disciplina
- 1 prova no último dia (nota única)
- Conceitos
  - 0 <= nota < 3, então E
  - 3 <= nota < 5, então D
  - 5 <= nota < 7, então C
  - 7 <= nota < 9, então B
  - 9 <= nota <= 10, então A

Conteúdo programático

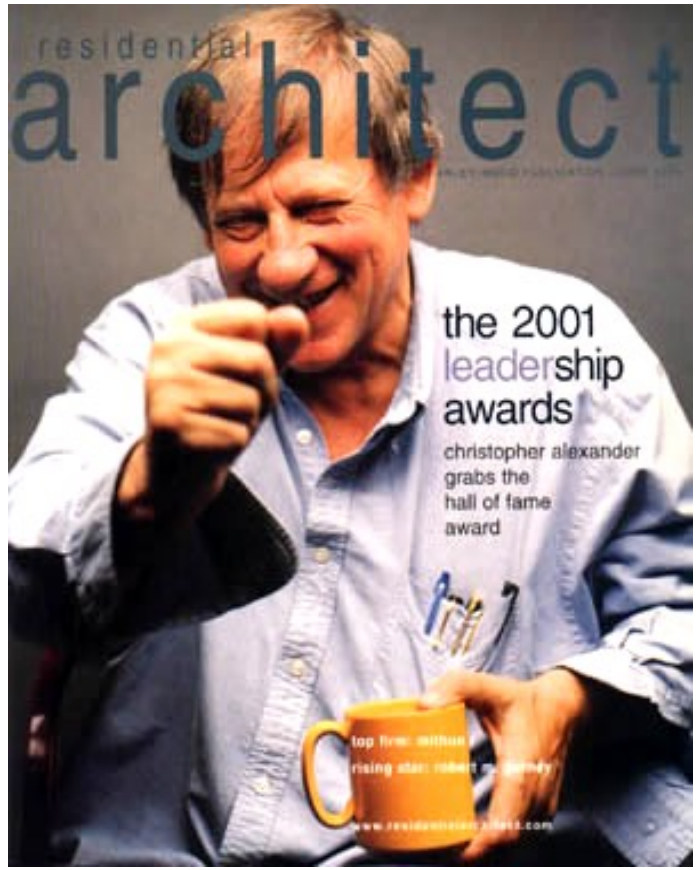
- Introdução
- Padrões **GRASP** (*General Responsibility Assignment Software Patterns*)
  - Caps. 17 e 25 do livro do Craig Larman
- Padrões **GoF** (*Gang of Four*)
  - **Todo** o livro do Gamma *et al.*

Introdução
Um pouco de história

- **Christopher Alexander**
  - *A Pattern Language: Towns, Buildings, Construtions* (1977)
- **Gamma et al.**
  - *Design Patterns: Elements of Reusable Object-Oriented Software* (1994)

Introdução
Um pouco de história

- Um padrão descreve
  - **problema** que ocorre repetidamente
  - **solução** para esse problema de forma que se possa reutilizar a solução



Introdução
Por quê usar padrões?

- Aprender com a **experiência** dos outros
- O **jargão** facilita a comunicação de princípios
- Melhora a **qualidade** do software
- Descreve **abstrações** de software
- Ajuda a **documentar** a arquitetura
- Captura as partes **essenciais** de forma compacta

Introdução
No entanto, padrões…

- não apresentam uma solução exata
- não resolvem todos os problemas de *design*
- não é exclusivo de orientação a objetos

Introdução
Como selecionar um padrão?

- Entenda as **soluções**
- Estude o **relacionamento** entre os padrões
- Estude as **similaridades** entre os padrões
- Conheça as principais causas de **retrabalho**
- Considere o que pode **mudar**

Padrões GRASP

- *Information Expert*
- *Creator*
- *Controller*
- *Low Coupling*
- *High Cohesion*
- *Polymorphism*
- *Pure Fabrication*
- *Indirection*
- *Protected Variations*

Padrões GRASP
Information Expert

- **Problema**
  - A quem atribuir a responsabilidade sobre fornecer/manter uma informação?
- **Solução**
  - Atribuir a responsabilidade ao especialista – a classe que tem a informação necessária

Padrões GRASP
Creator

- **Problema**
  - Quem deve ser responsável pela criação de objetos?
- **Solução:** B deve ser *Creator* de A se
  - B agrega objetos de A
  - B contém objetos de A
  - B registra objetos de A
  - B usa de maneira muito próxima objetos de A
  - B tem os dados de inicialização de A

Padrões GRASP
Controller

- **Problema**
  - Quem deve ser responsável pelo controlar os eventos do sistema?
- **Solução:** uma classe que representa
  - O sistema como um todo
  - Todo o negócio/organização
  - Algo no mundo real envolvido na tarefa

Padrões GRASP
Controller – Facade Controller

- Centraliza acesso ao sistema em um controlador
- **Prós**
  - Centralizado
  - Baixo acoplamento na interface
  - Simplicidade
- **Contras**
  - Centralizado
  - *Controller* tem acoplamento alto
  - Coesão baixa

Padrões GRASP
Controller – Role Controller

- Divide acesso ao sistema de acordo com o papel dos usuários ou sistemas externos
- **Prós**
  - Coesão melhora
  - Descentralizado
  - Reduz acoplamento do Controller
- **Contras**
  - Pode ser mal balanceado

Padrões GRASP
Controller – Use Case Controller

- Divide acesso ao sistema de acordo os casos de uso do sistema
- **Prós**
  - Coesão
- **Contras**
  - Acoplamento aumenta
  - Explosão de classes

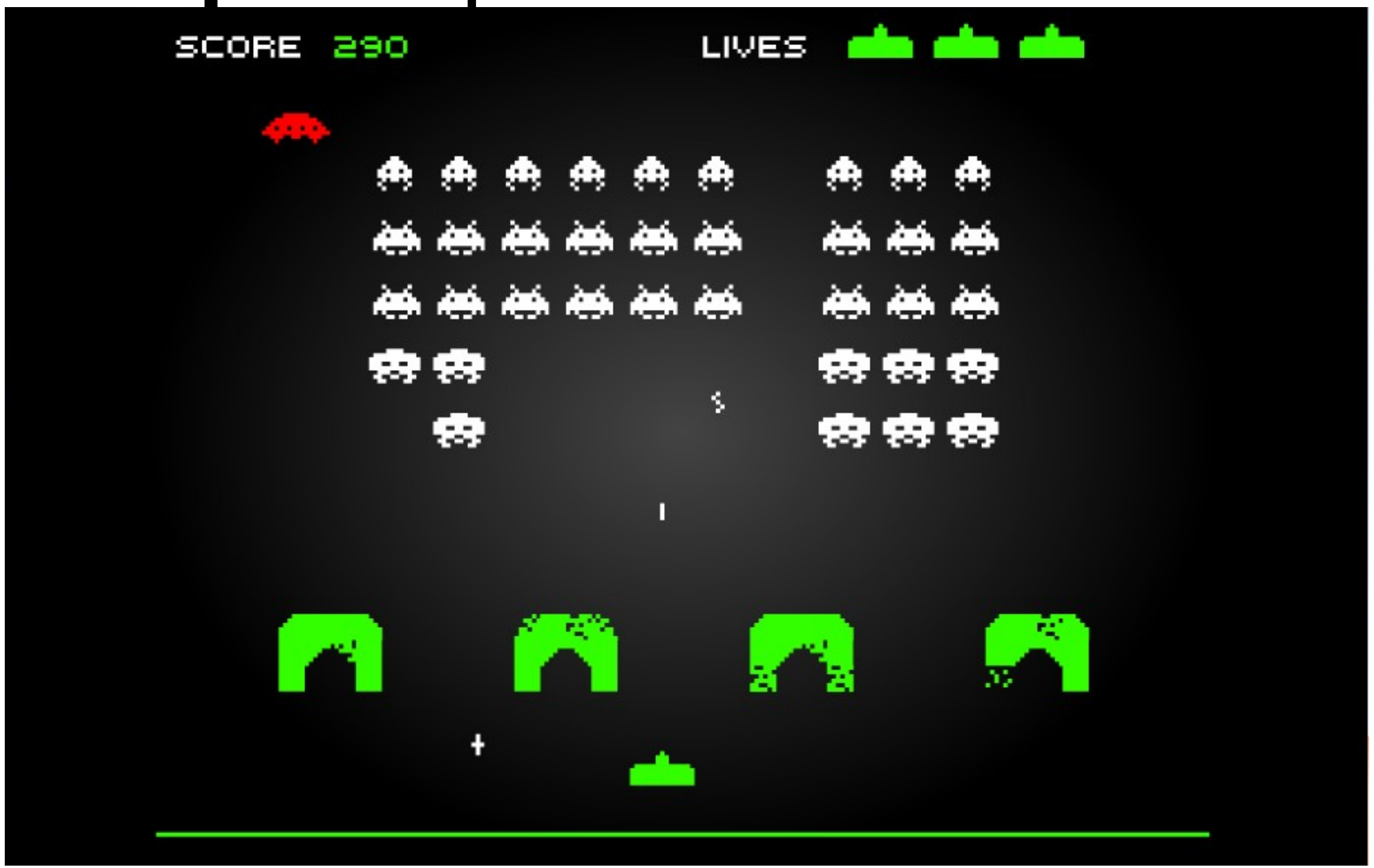
Padrões GRASP
Low Coupling

- O acoplamento (dependência entre classes) deve ser mantido o mais baixo possível
- Como medir acoplamento?
  - Representar o diagrama de classes como um dígrafo e computar o grau de saída
  - Métrica CK – *Coupling Between Objects* (CBO)

Padrões GRASP
High Cohesion

- A coesão (grau de “relacionamento” entre as funcionalidades de uma classe) deve ser mantida alta
- Como medir coesão?
  - Acúmulo de responsabilidades
  - Coesão entre nome de classes e métodos
  - Métrica CK – *Lack of Cohesion On Methods* (LCOM)

Padrões GRASP
Exemplo: Space Invaders



Fonte: http://www.freespaceinvaders.org/

Padrões GRASP
Exemplo: Space Invaders

- Quem é **responsável** por
  - Manter as coordenadas dos alienígenas?
  - Manter as coordenadas das naves?
  - Criar novas naves no início das fases?
  - Controlar pontuação?
  - Controlar movimento dos personagens?
  - Lançar tiros?

Padrões GRASP
Polymorphism

- **Problema**
  - Quem é responsável quando comportamento varia?
- **Solução**
  - Quando comportamentos relacionados variam, use operações polimórficas nos tipos onde o comportamento varia

Padrões GRASP
Pure Fabrication

- **Problema**
  - Quem é responsável quando (você está desesperado e) não quer violar alta coesão e baixo acoplamento?
- **Solução**
  - Associe um conjunto coeso de responsabilidades para uma classe “artificial” (que não representa um conceito do domínio do problema) tento em vista alta coesão, baixo acoplamento e reúso

Padrões GRASP
Indirection

- **Problema**
  - Como associar responsabilidades para evitar acoplamento direto?
- **Solução**
  - Use um objeto intermediário que faça mediação entre outros componentes/serviços

Padrões GRASP
Protected Variations

- Princípio fundamental de ***Software Design***
- **Problema**
  - Como projetar sistemas de tal forma que variações ou instabilidade não tenham impacto indesejado em outros elementos?
- **Solução**
  - Identificar pontos de variação/instabilidade e usar uma interface (no sentido mais amplo) estável como ponto de acesso

Métricas CK

- WMC (*Weighted Methods per Class*)
- DIT *(Depth of Inheritance Tree)*
- NOC *(Number of Children)*
- **CBO *(Coupling Between Objects)***
- RFC *(Response for a Class)*
- **LCOM *(Lack of Cohesion in Methods)***

Métricas CK
CBO

- Número de classes com a qual uma determinada classe está acoplada
- Classes estão acopladas se
  - métodos de uma classe usam
    - métodos de outra classe
    - atributos de outra classe

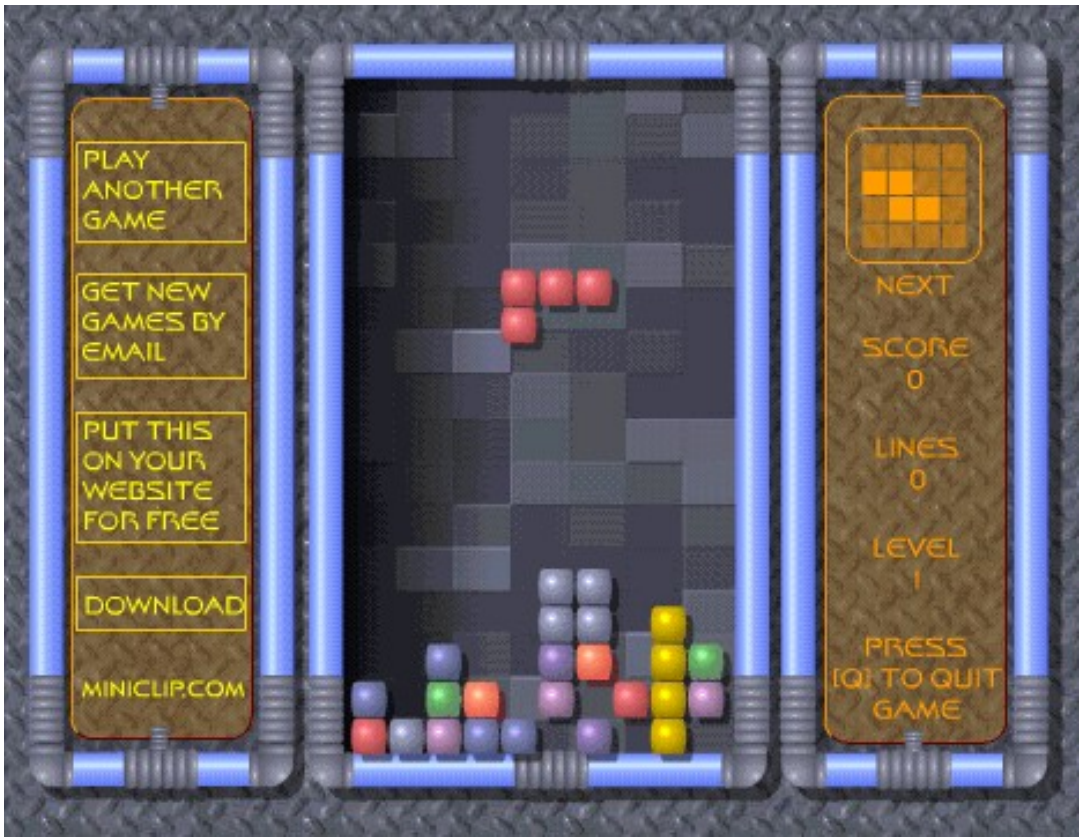
Métricas CK
LCOM

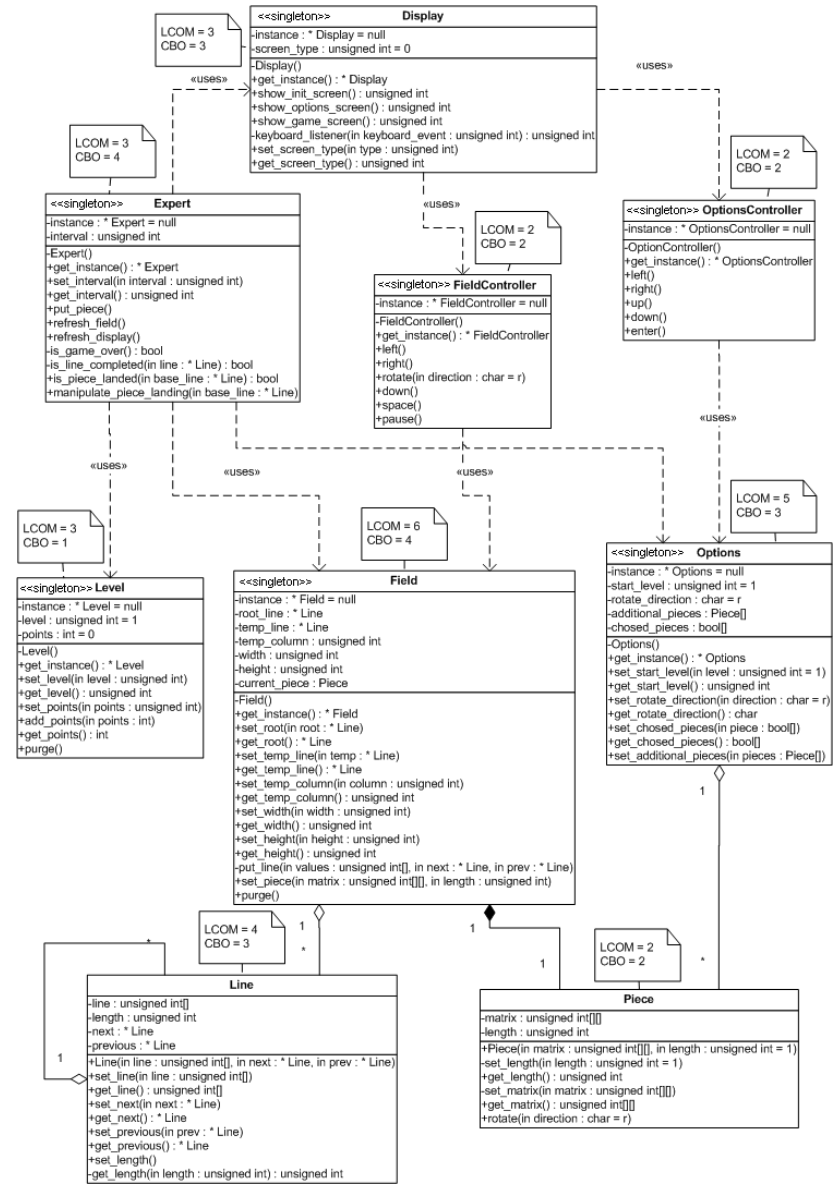
- Número de interseções ***vazias*** entre métodos de uma classe
  - considerando variáveis usadas
- Quanto mais métodos se “parecem” mais coesa é a classe
- Em outras palavras, conta quantos pares de métodos não têm “nada a ver”

Métricas CK
LCOM

- **Exemplo:**
  - Considere uma Classe C com métodos
    - M1 usando variáveis {V1} = {*a, b, c, d, e*}
    - M2 usando variáveis {V2} = {*a, b, e*}
    - M3 usando variáveis {V3} = {*x, y, z*}
  - E interseção entre {V1}, {V2} e {V3}:
    - {V1} ∩ {V2} = {a, b, e}
    - {V1} ∩ {V3} = {}
    - {V2} ∩ {V3} = {}
  - Então LCOM de C = 2

Padrões GRASP
Exemplo: Tetris





GoF – Criacionais (Capítulo 3)

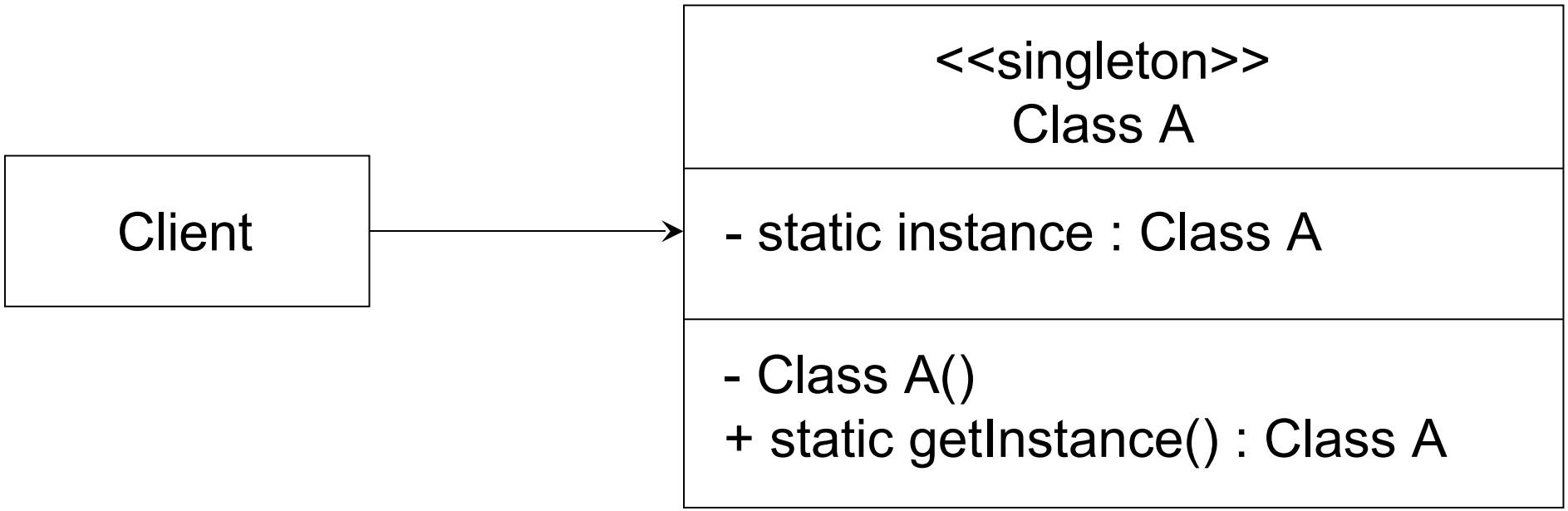
- *Singleton*
- *Factory Method*
- *Abstract Factory*
- *Prototype*
- *Builder*

GoF – Criacionais
Singleton

- **Intenção:** Garantir que uma classe tenha somente uma instância e fornecer um ponto de acesso à instancia

GoF – Criacionais
Singleton

- **Estrutura:**



GoF – Criacionais
Singleton

- **Exemplo:**

```
class Singleton{
  private static Singleton instance;
  private Singleton{ }
  public static Singleton getInstance(){
    if( instance == null )
      instance = new Singleton();
    return instance;
  }
}
```

GoF – Criacionais
Singleton

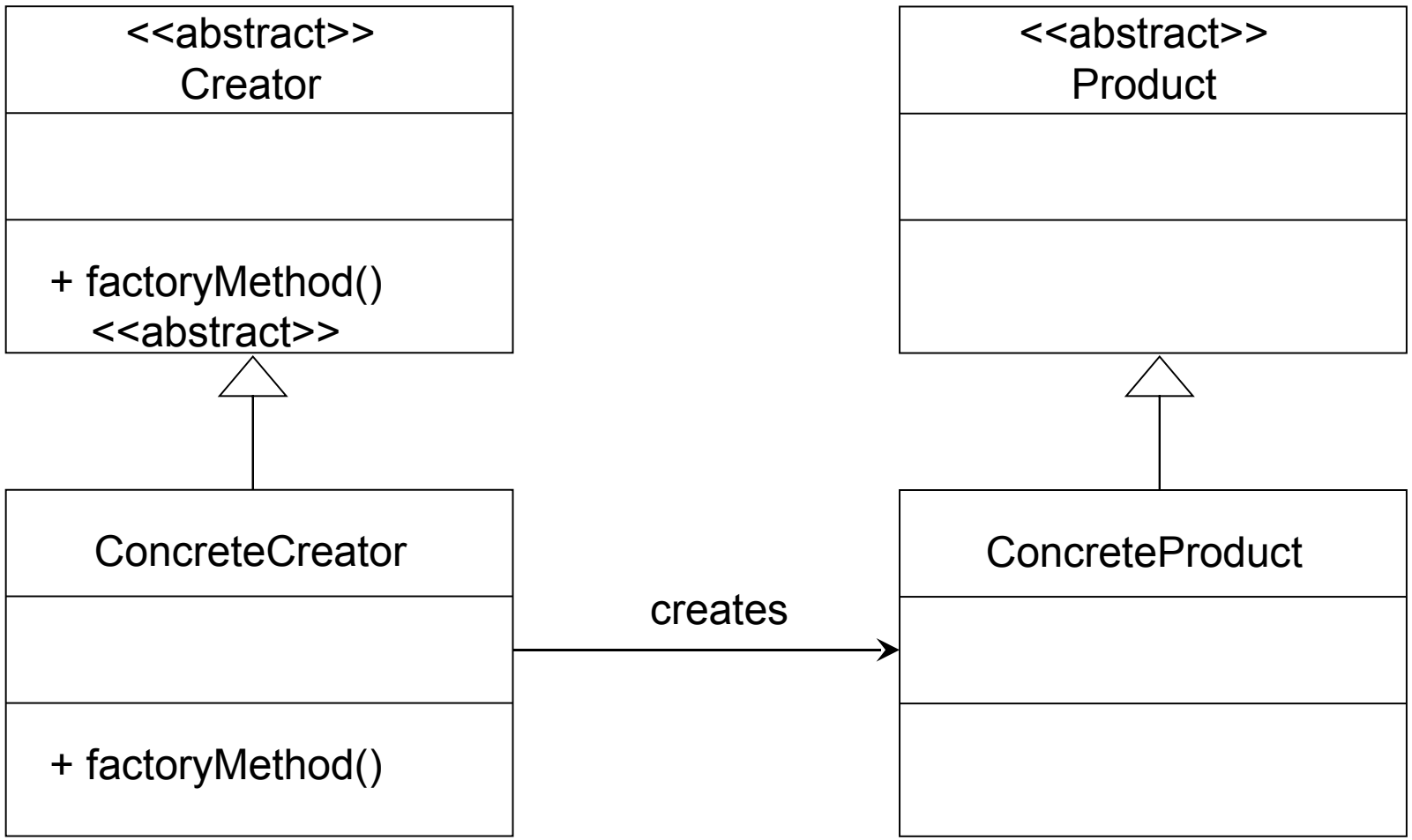
- **Use quando:**
  - Deve haver exatamente uma instância da classe
  - Deve deve ser facilmente acessível aos clientes em um ponto de acesso bem conhecido

GoF – Criacionais
Factory Method

- **Intenção:** Definir uma interface para criação de um objeto, mas deixar as subclasses decidirem qual classe instanciar

GoF – Criacionais
Factory Method

- **Estrutura:**



GoF – Criacionais
Factory Method

- **Exemplo:**

```
abstract class Product{
   ...
}

class ConcreteProductA extends Product{
   ...
}

class ConcreteProductB extends Product{
   ...
}
```

GoF – Criacionais
Factory Method

- **Exemplo:**

```
abstract class Creator{
  public abstract Product create();
}

class ConcreteCreatorA extends Creator{
  public Product create(){
    return new ConcreteProductA() ;
 }
}

class ConcreteCreatorB extends Creator{
  public Product create(){
    return new ConcreteProductB() ;
 }
}
```

GoF – Criacionais
Factory Method

- **Exemplo:**

```
class GoFTest{
  public static void main( String a[] ){
    Creator c ;
    // If A is needed
    c = new ConcreteCreatorA() ;
    // else
    c = new ConcreteCreatorB() ;
    Product p = c.create() ;
  }
}
```

GoF – Criacionais
Factory Method

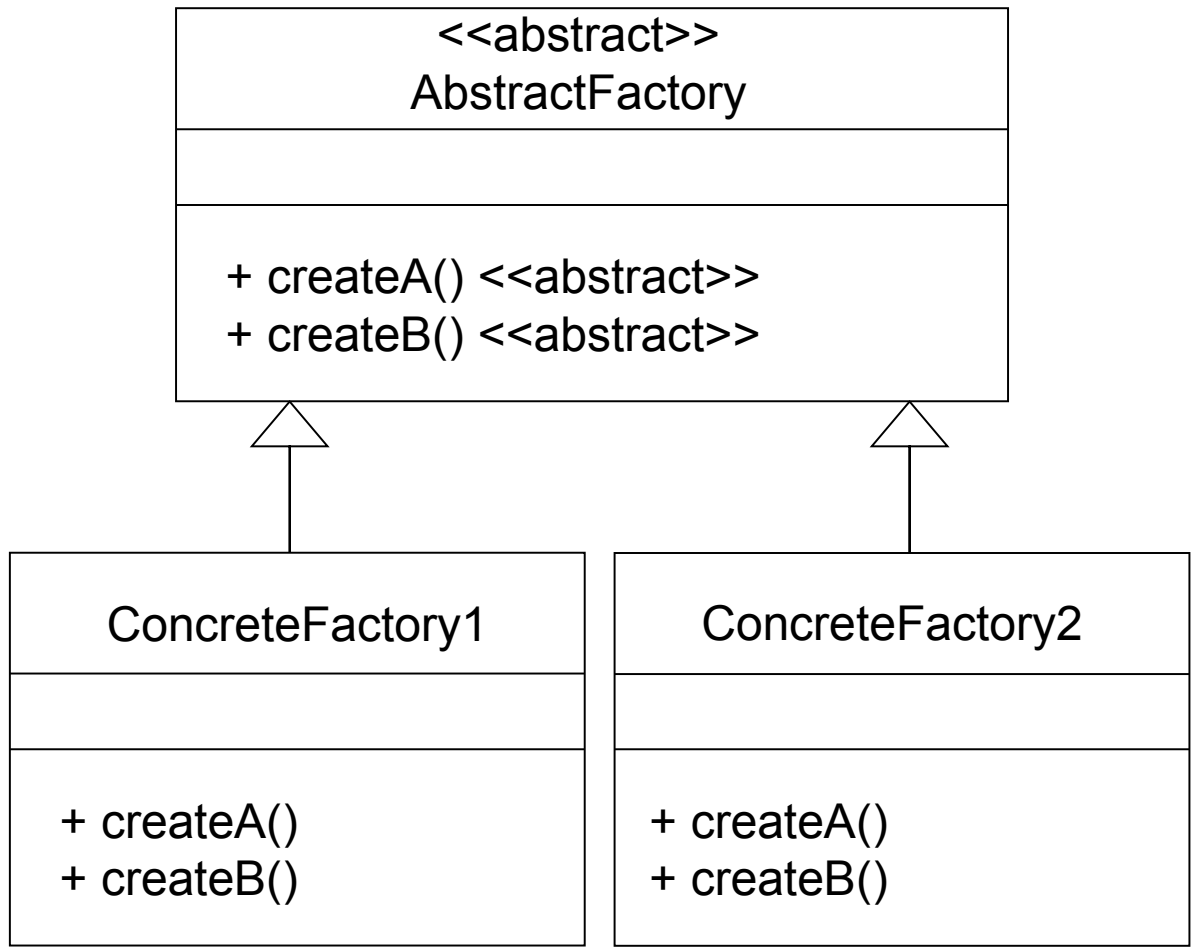
- **Use quando:**
  - Uma classe não pode antecipar a classe de objetos que precisa criar
  - Uma classe deseja que suas subclasses especifiquem os objetos que cria

GoF – Criacionais
Abstract Factory

- **Intenção:** Fornecer interface para criação de **famílias** de objetos relacionados ou dependentes sem especificar suas classes concretas

GoF – Criacionais
Abstract Factory

- **Estrutura:**



GoF – Criacionais
Abstract Factory

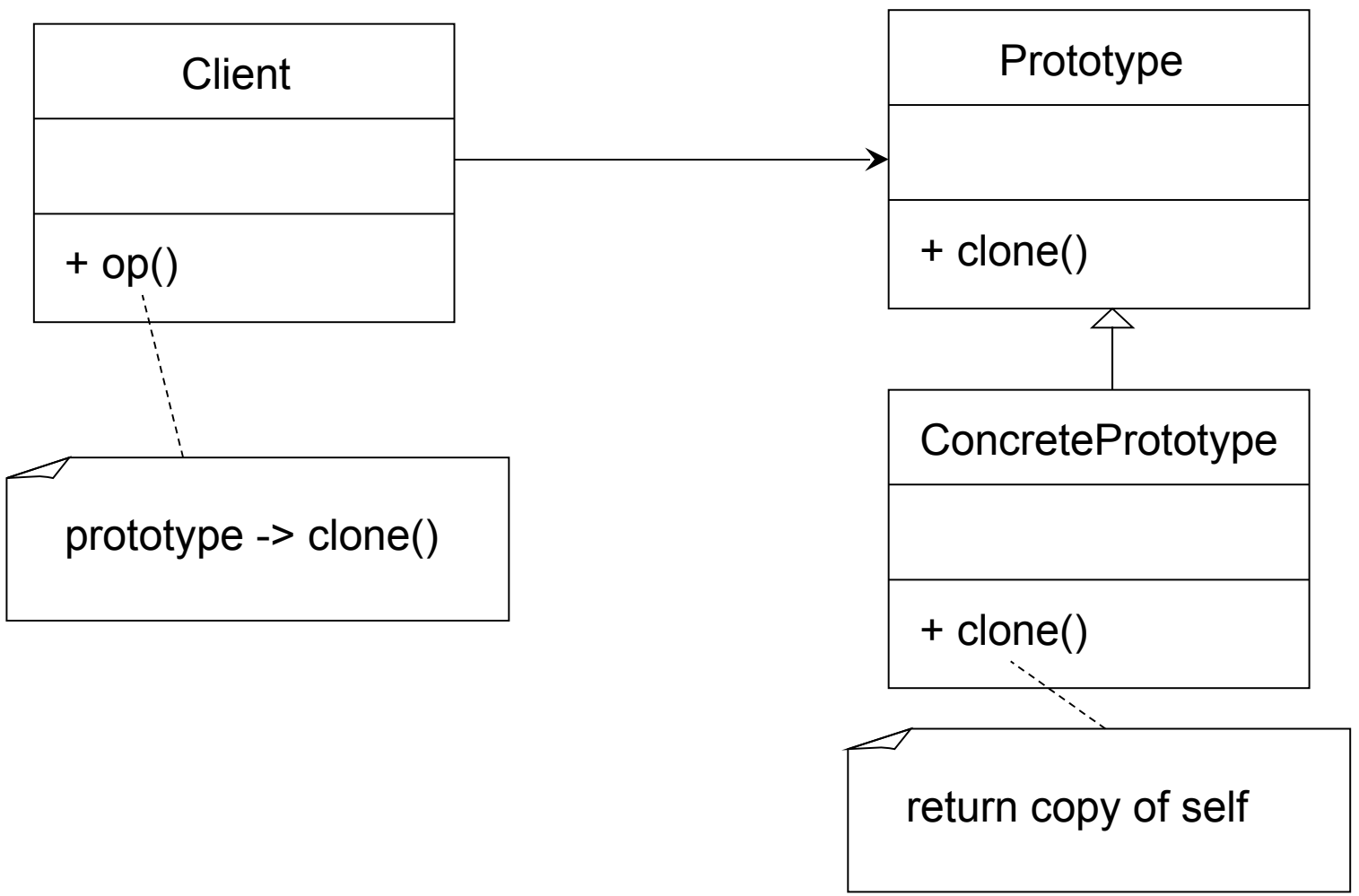
- **Use quando:**
  - Um sistema deveria ser independente de como seus produtos são criados, compostos e representados
  - Um sistema deveria ser configurados com uma ou várias famílias de produtos
  - Uma família de objetos é destinada a ser usada de maneira única

GoF – Criacionais
Prototype

- **Intenção:** Especificar os tipos de objetos a serem criados usando uma instância-protótipo. Novos objetos são criados pela cópia desse protótipo

GoF – Criacionais
Prototype

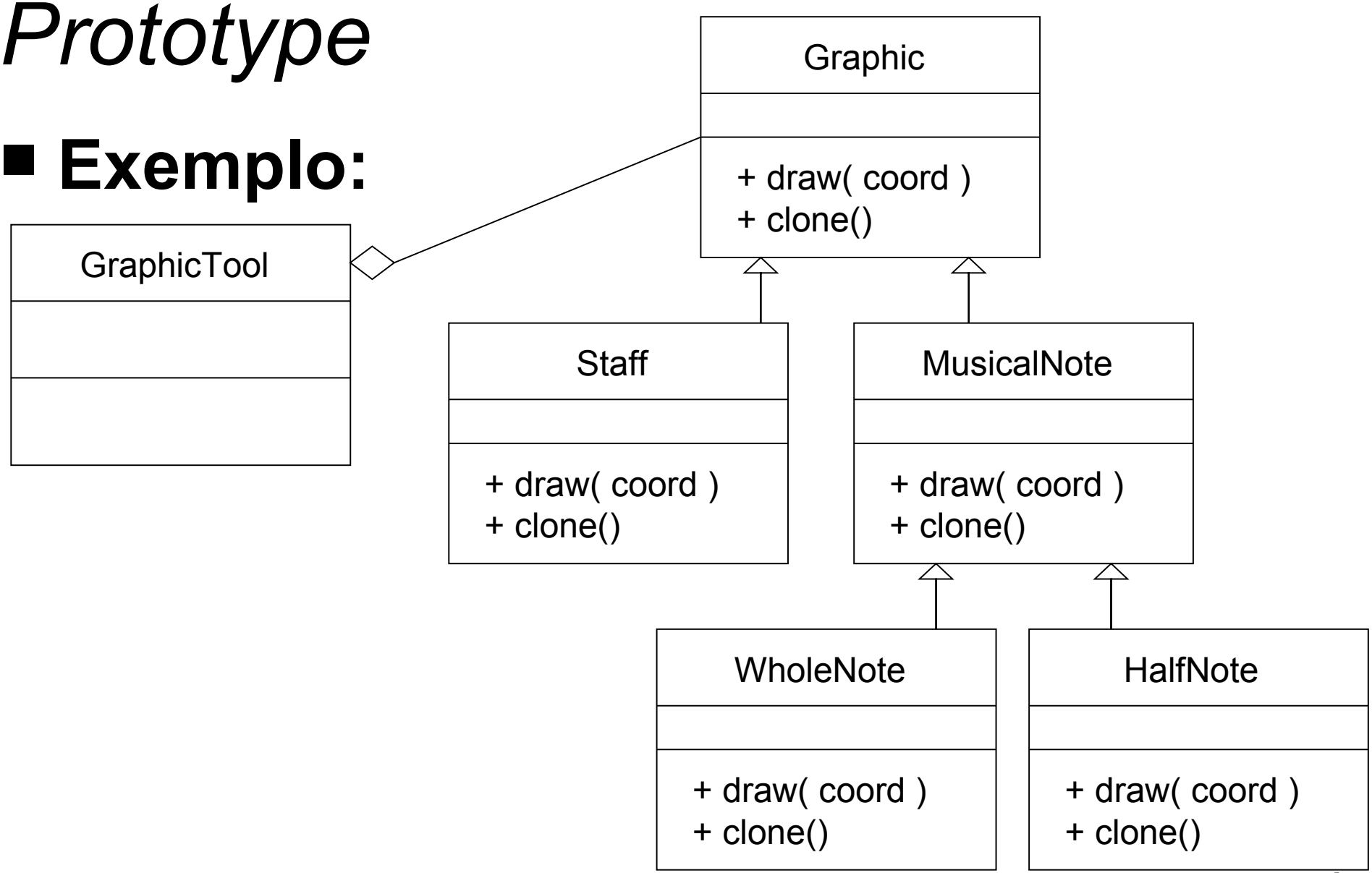
- **Estrutura:**



GoF – Criacionais

Prototype

- **Exemplo:**



GoF – Criacionais
Prototype

- **Use quando:**
  - Classes a instanciar são especificadas em tempo de execução
  - Instâncias de classes podem ter poucas combinações de estado

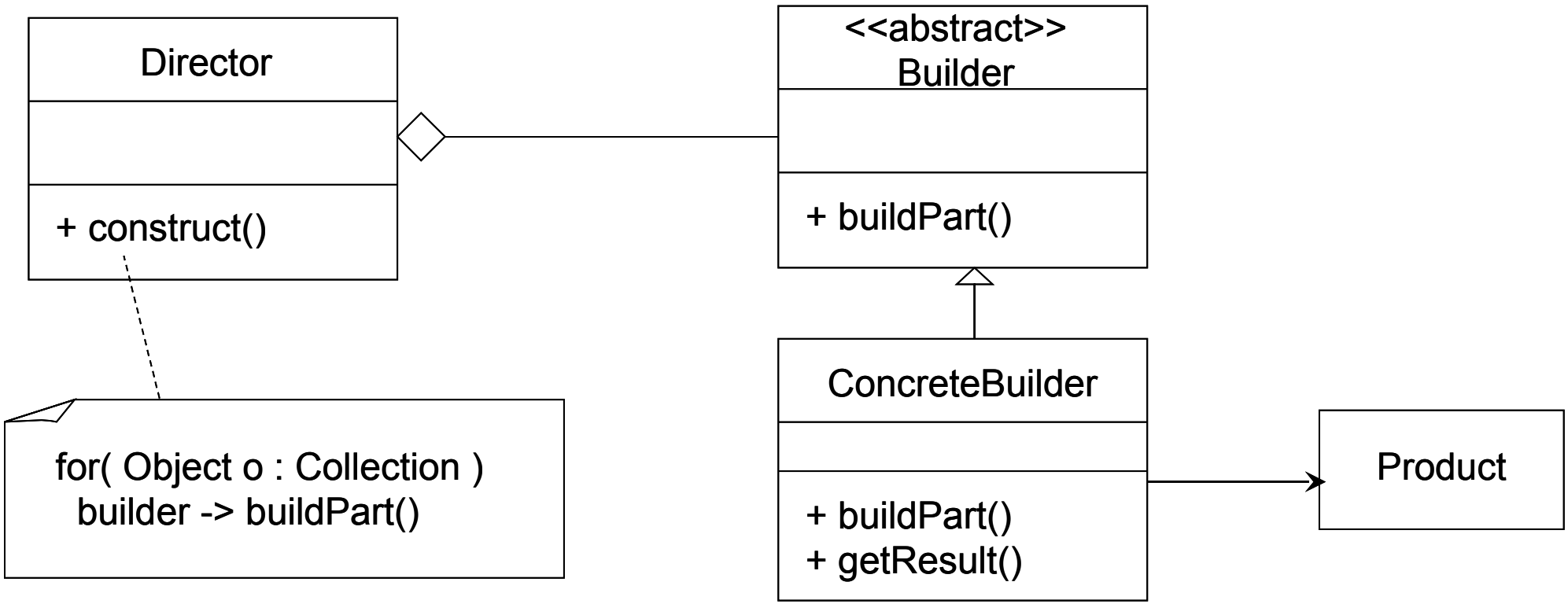
GoF – Criacionais
Builder

- **Intenção:** Separar a construção de um objeto complexo de sua representação de modo que o mesmo processo de construção possa criar diferentes representações

GoF – Criacionais

Builder

- **Estrutura:**



GoF – Criacionais
Builder

- **Exemplo:**

```
class Director{
   ...
   private Builder b ;
   b = new ConcreteBuilder() ;
   for( Object o : Collection )
     b.buildPart( o ) ;
   Product p = b.getResult() ;
   ...
}
```

GoF – Criacionais

Builder

- **Exemplo:**

```
abstract class Builder{
  abstract buildPart( Part p );
}

class ConcreteBuilder extends Builder{
  public Part buildPart( PartA a ){...}
  public Part buildPart( PartB b ){...}
  public Product getResult(){...}
  // Product of A&B is returned
}
```

GoF – Criacionais
Builder

- **Use quando:**
  - O algoritmo para criar um objeto complexo deveria ser independente das partes que compõem o objeto
  - O processo de construção deve permitir diferentes representações para o objeto que é construídos

Exercício

- Em duplas ou trios
- Aplicar padrões vistos até o momento em
  - Projeto para “Radar” de fiscalização de velocidade
    - Componentes a considerar
      - Sensor de presença
      - Máquina fotográfica
      - Central
      - Outros (?)

GoF – Estruturais (Capítulo 4)

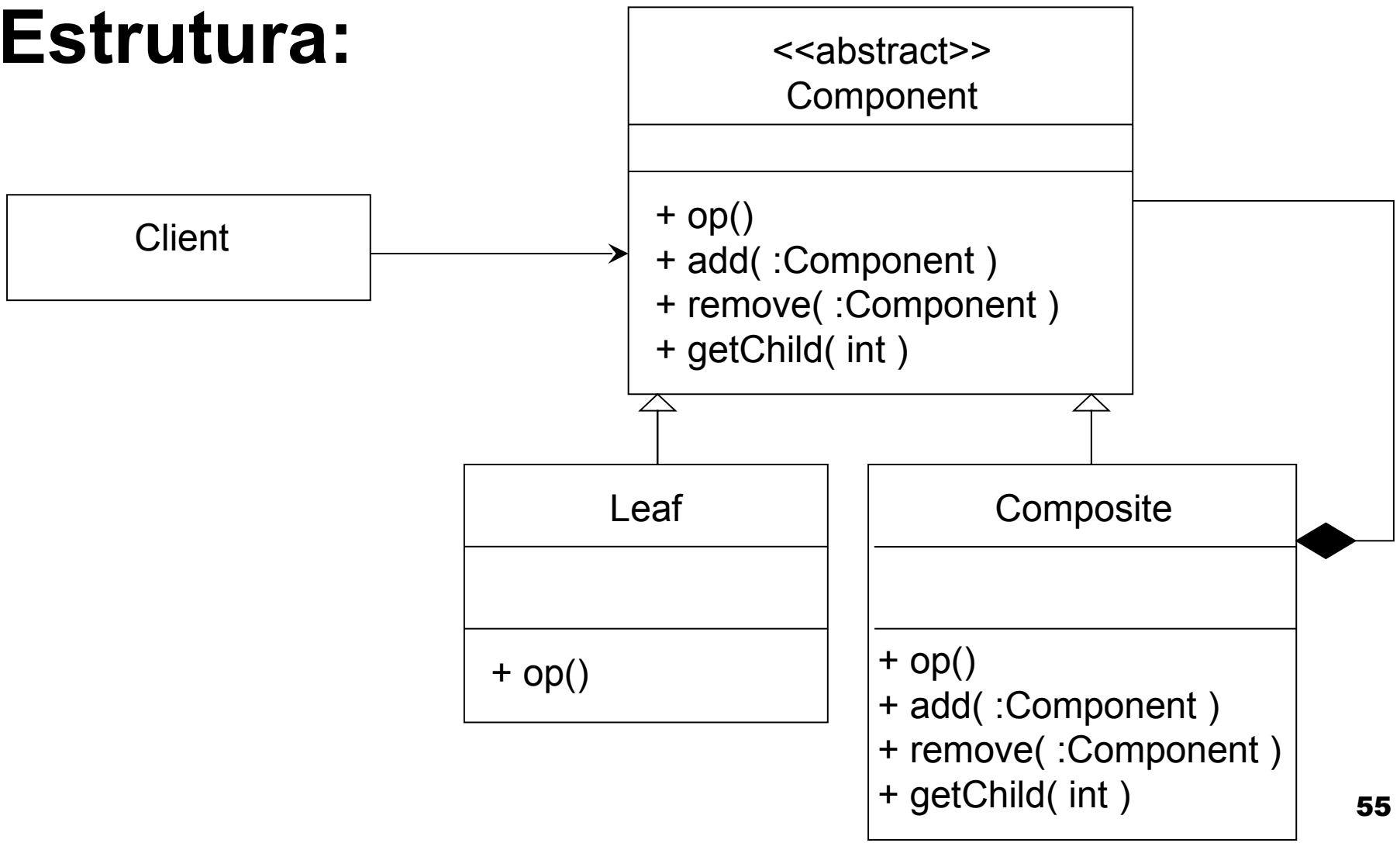
- *Composite*
- *Decorator*
- *Proxy*
- *Adapter*
- *Bridge*
- *Facade*
- *Flyweight*

GoF – Estruturais
Composite

- **Intenção:** Compor objetos em estruturas de árvore para representarem hierarquias do tipo todo-parte

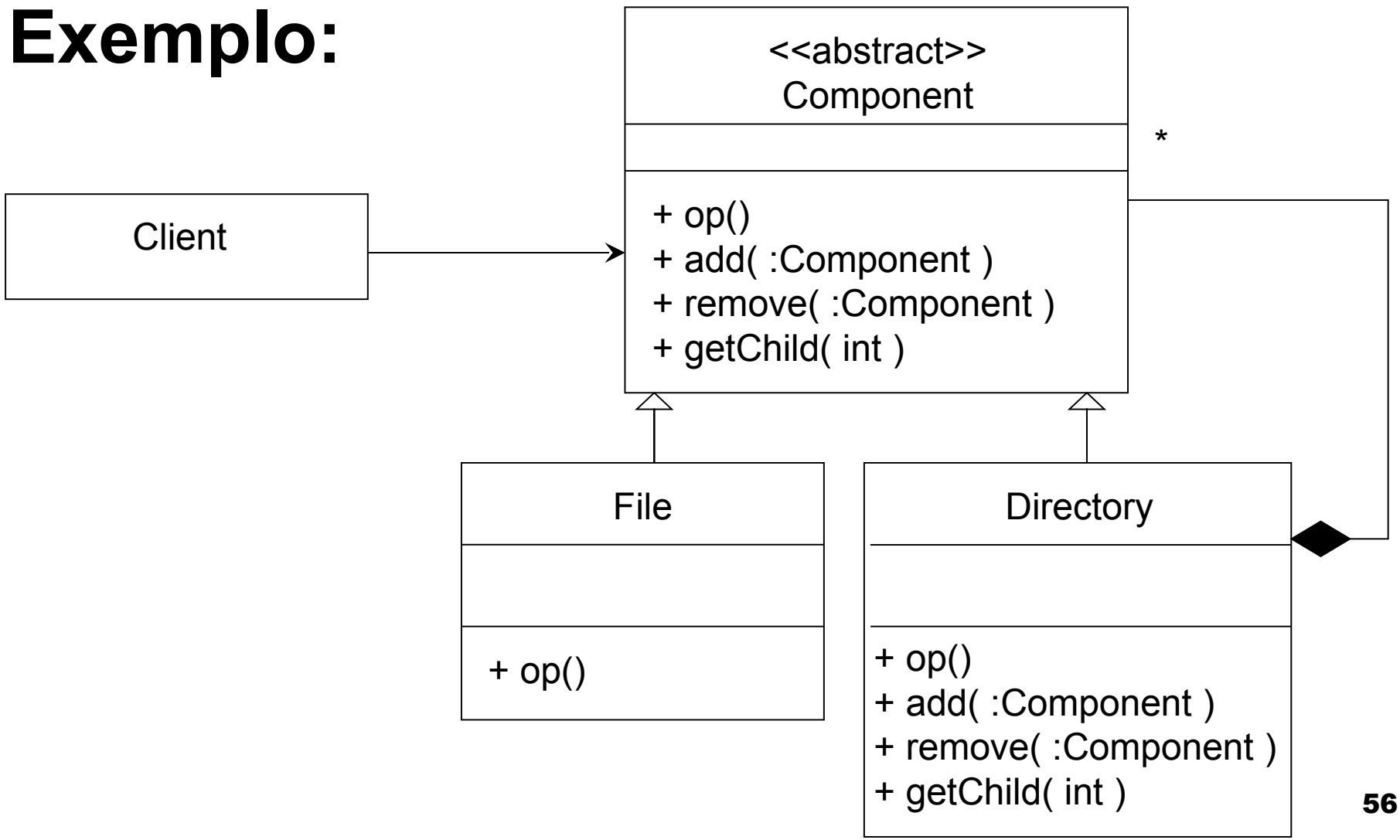
GoF – Estruturais
Composite

- **Estrutura:**



GoF – Estruturais
Composite

- **Exemplo:**



GoF – Estruturais
Composite

- **Use quando:**
  - Você quer representar hierarquia de objetos do tipo parte-todo
  - Você quer que clientes tratem objetos compostos e individuais da mesma forma

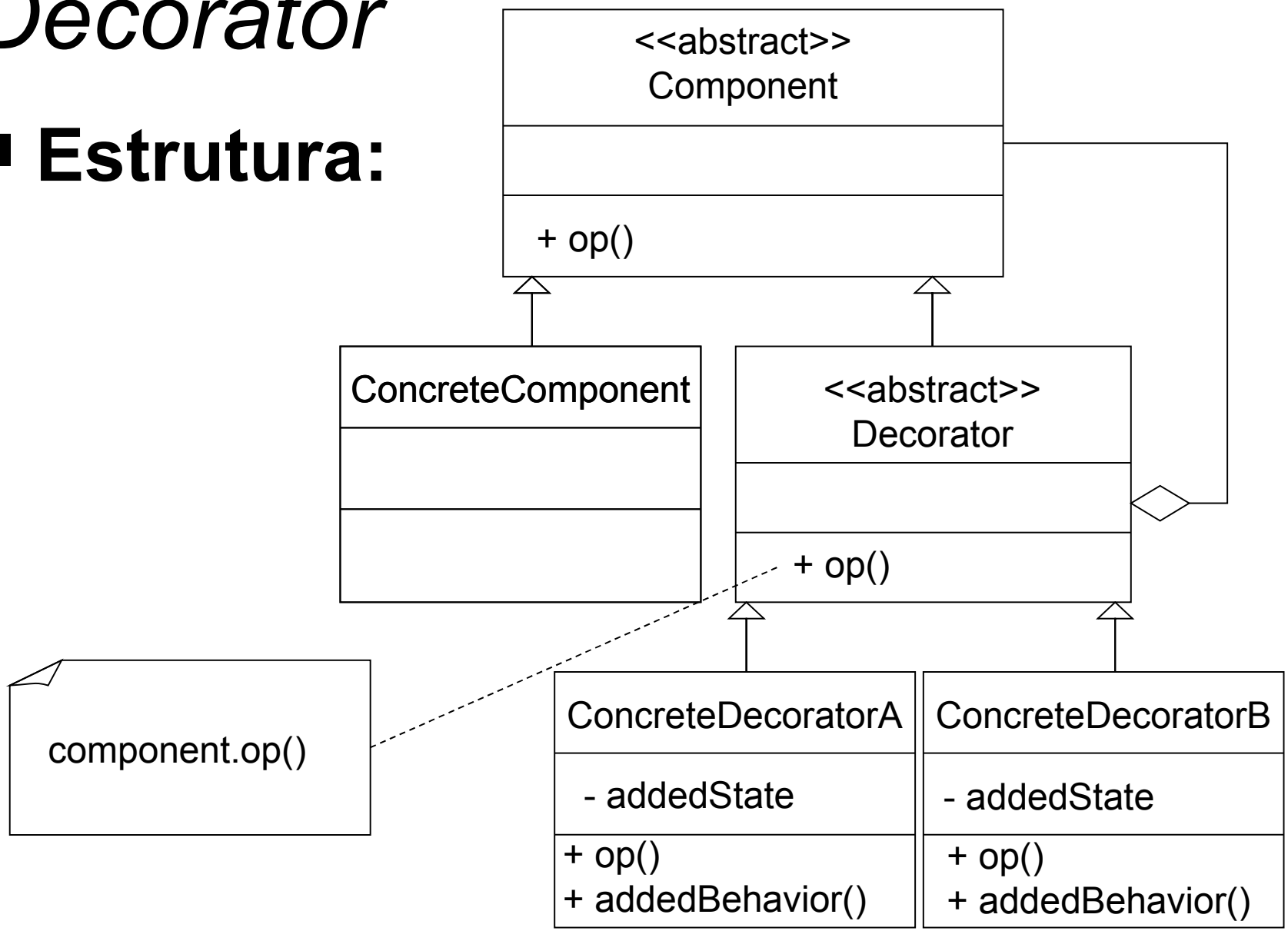
GoF – Estruturais
Decorator

- **Intenção:** Dinamicamente, agregar funcionalidades a um objeto

GoF – Estruturais

Decorator

- **Estrutura:**



GoF – Estruturais
Decorator

- **Exemplo:**

```
abstract class Decorator{
   ...
   private Component component ;
   public Decorator( Component c ){
      component = c ;
   }
   ...
   public void operation(){
      component.operation() ;
   }
}
```

GoF – Estruturais
Decorator

- **Exemplo:**

```
class GoFTest{
   ...
   Component c = new ConcreteDecoratorA(
     new ConcreteDecoratorB(
       new ConcreteComponent()));
   c.operation();
   ...
}
```

GoF – Estruturais
Decorator

- **Outro exemplo:**

```
Sanduich s = new Hamburguer(
 new Hamburguer(
  new Letuce(
   new Cheese(
    new SpecialSpice(
     new Onion(
      new Pickles(
       new BreadWithGergelim()))))));
```

GoF – Estruturais
Decorator

- **Use quando:**
  - Deseja adicionar responsabilidades para objetos individuais dinamicamente, de maneira transparente e sem afetar outros objetos
  - Quando uma hierarquia de subclasses não é prática devido ao grande número de possibilidades (explosão de classes)

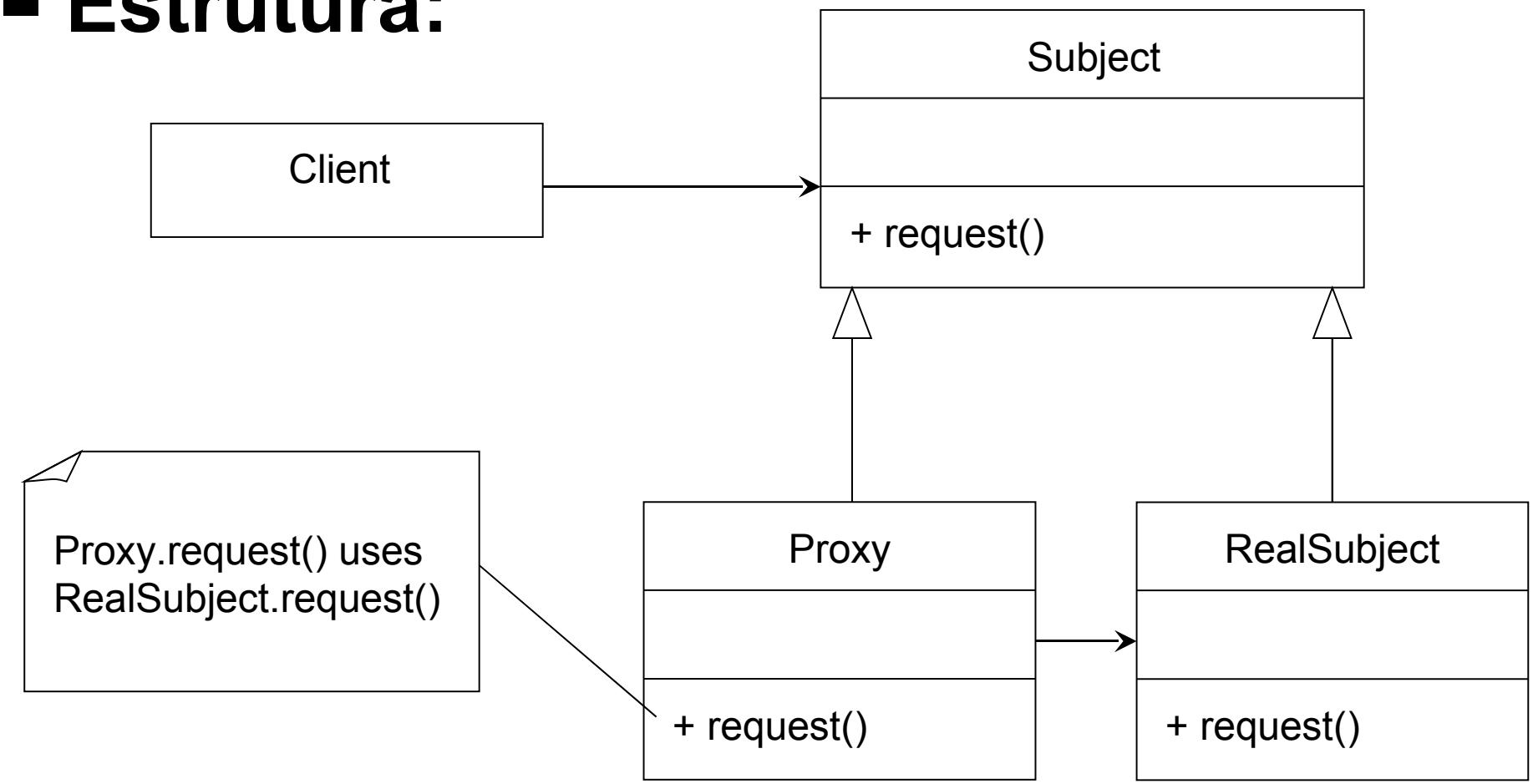
GoF – Estruturais
Proxy

- **Intenção:** Fornecer um representante de um objeto para controlar o acesso ao mesmo

GoF – Estruturais

Proxy

- **Estrutura:**



GoF – Estruturais
Proxy

- **Exemplo:**

```
class RealSubject extends Subject{
   ...
   public request(){
     // implementation of the request
   }
   ...
}
```

GoF – Estruturais
Proxy

- **Exemplo:**

```
class Proxy extends Subject{
   ...
   public request(){
      Subject s = new RealSubject() ;
      s.request() ;
   }
   ...
}
```

GoF – Estruturais
Proxy

- **Exemplo:**

```
class GoFTest{
   ...
   Subject s = new Proxy() ;
   s.request() ;
   ...
}
```

GoF – Estruturais
Proxy

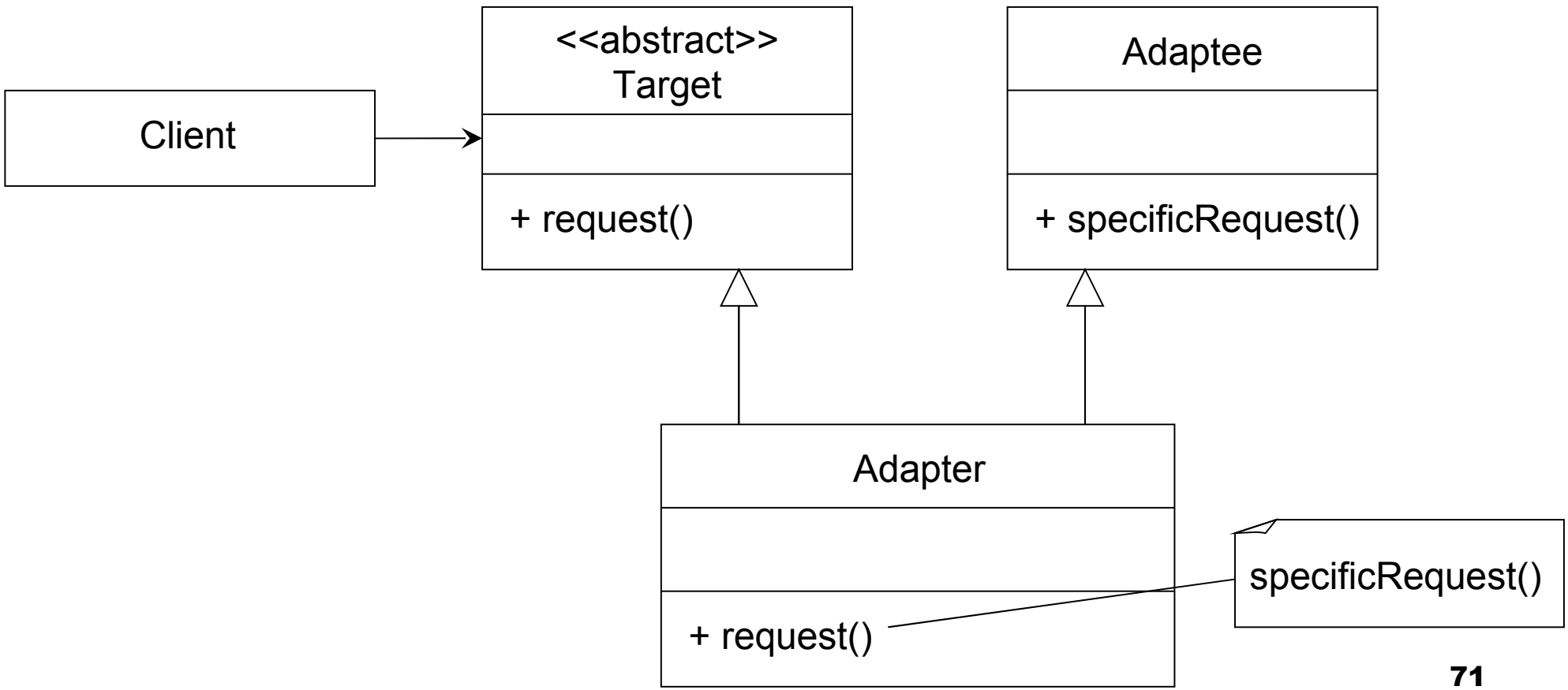
- **Use quando:**
  - Há a necessidade de uma referência sofisticada ou versátil a um objeto (mais do que um simples ponteiro)

GoF – Estruturais
Adapter

- **Intenção:** Converter a interface de uma classe em outra que os clientes esperam

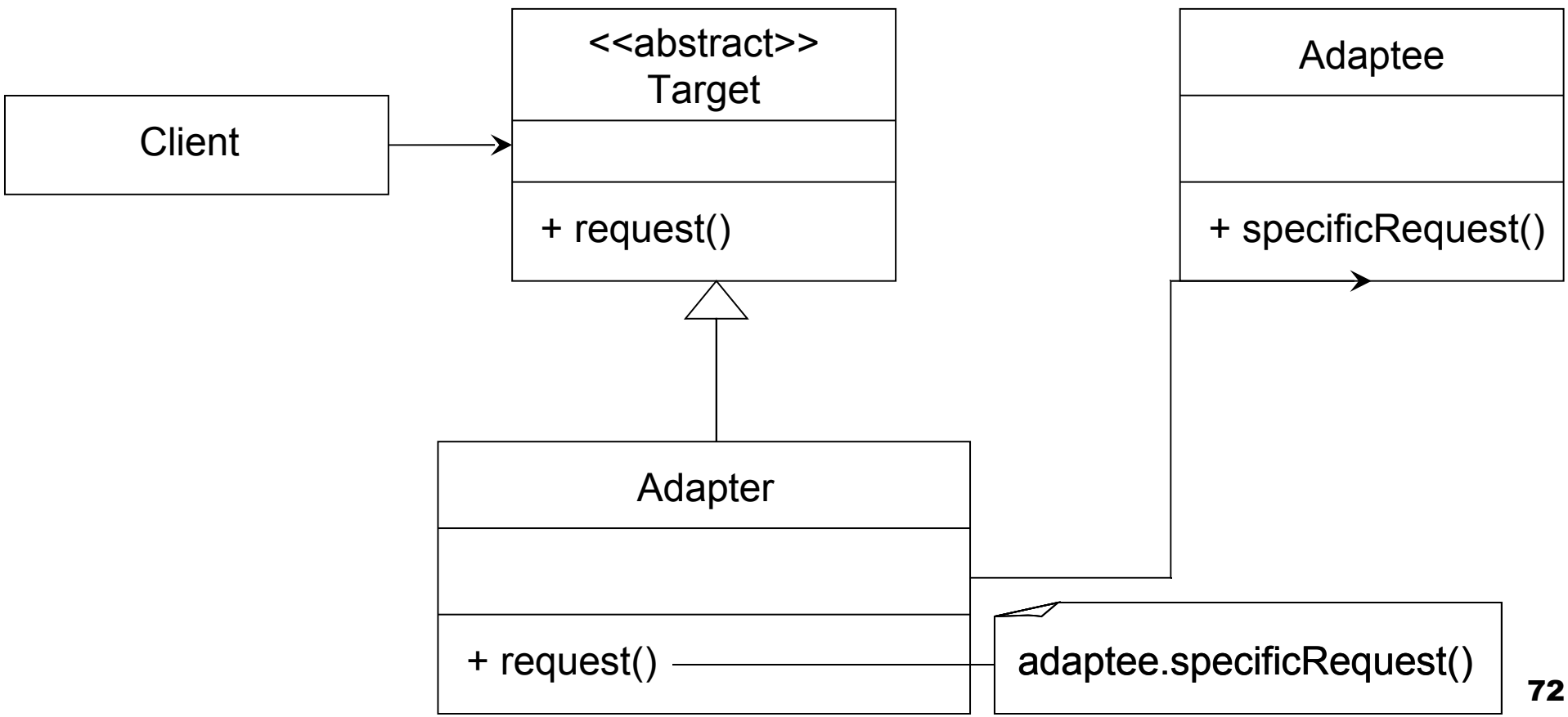
GoF – Estruturais
Adapter

- **Estrutura (*class adapter*):**



GoF – Estruturais
Adapter

- **Estrutura (*object adapter*):**



GoF – Estruturais

Adapter

- **Exemplo (*object adapter*):**

```
abstract class Target{
  public abstract void request();
}
class Adapter extends Target{
  public void request(){
    Adaptee a = new Adaptee();
    a.specificRequest();
  }
}
```

GoF – Estruturais
Adapter

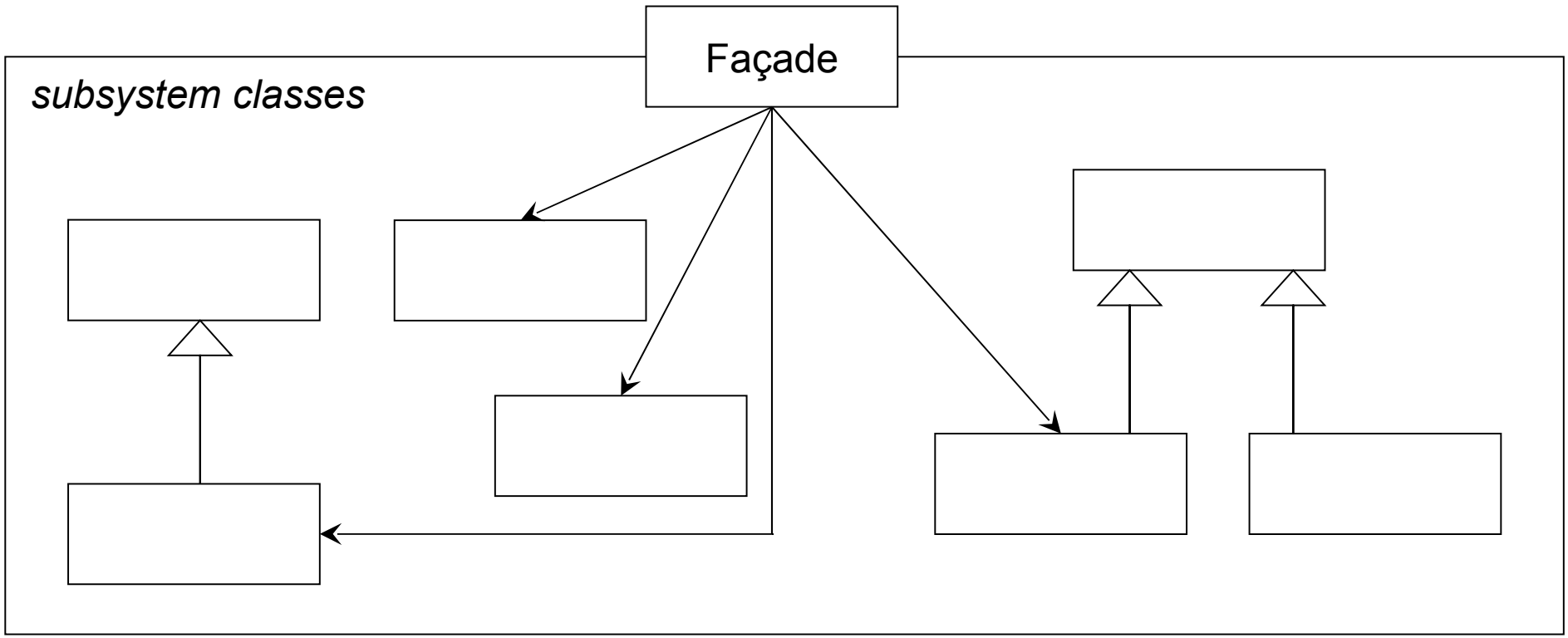
- **Use quando:**
  - Deseja usar uma classe existente, mas sua interface não combina com o que precisa
  - Você precisa criar classes reutilizáveis que cooperem com classes não previstas

GoF – Estruturais
Facade

- **Intenção:** Fornecer uma interface unificada para um conjunto de interfaces de um subsistema

GoF – Estruturais
Facade

- **Estrutura:**



GoF – Estruturais
Facade

- **Exemplo:**

```
class Facade{
  public Response parseRequest(
      Request r ){
    RequestController rc;
    rc = RequestController.getInstance();
    return rc.parse( r );
  }
  public boolean areYouAlive(){
    SystemController sc ;
    sc = SystemController.getInstance()  ;
    return sc.isAlive();
  }
}
```

GoF – Estruturais
Facade

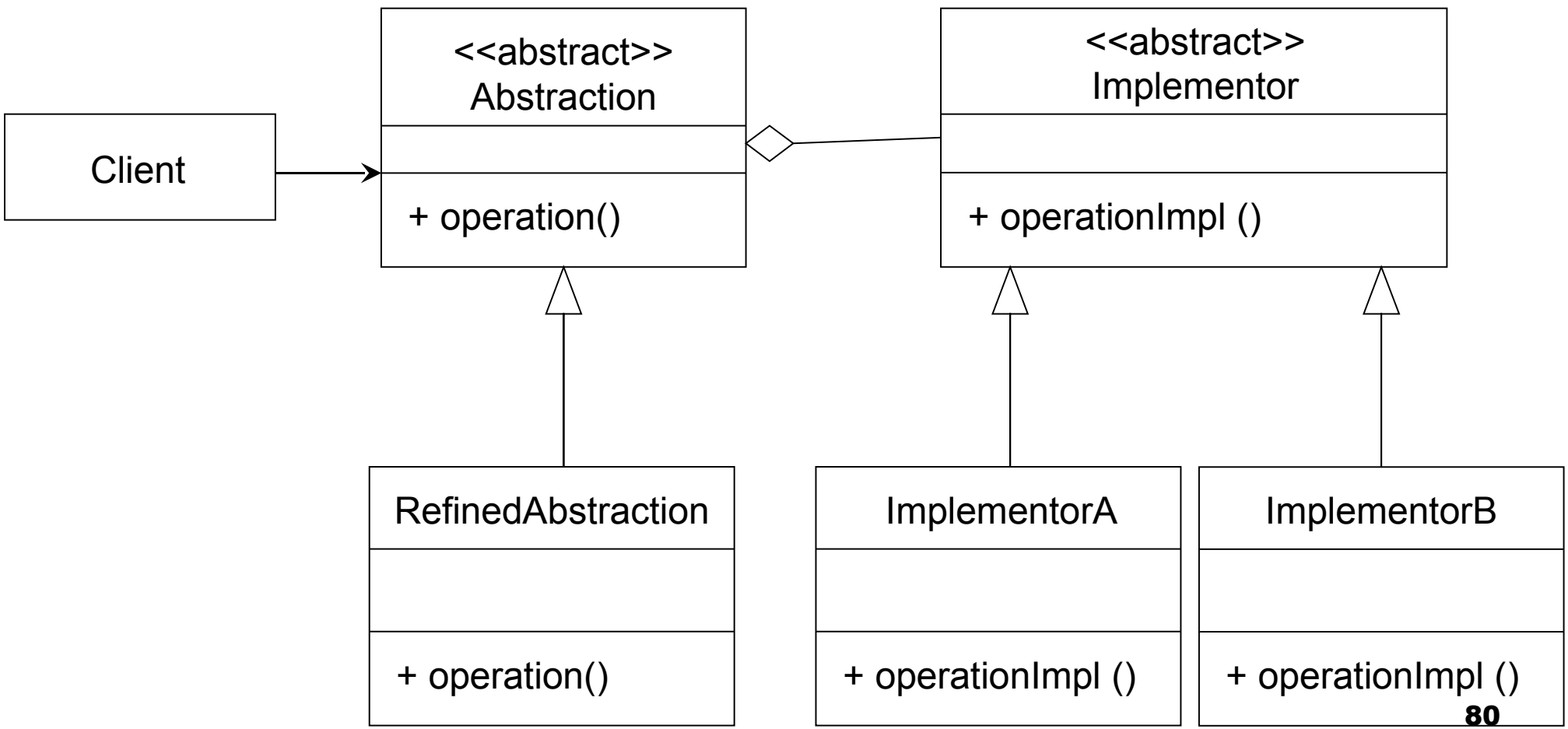
- **Use quando:**
  - Precisar de uma interface simples para um subsistema complexo
  - Há muitas dependências entre clientes e classes de implementações de uma abstração
  - Desejar dividir seu sistema em camadas

GoF – Estruturais
Bridge

- **Intenção:** Desacoplar uma abstração de sua implementação, de modo que as duas possam variar independentemente

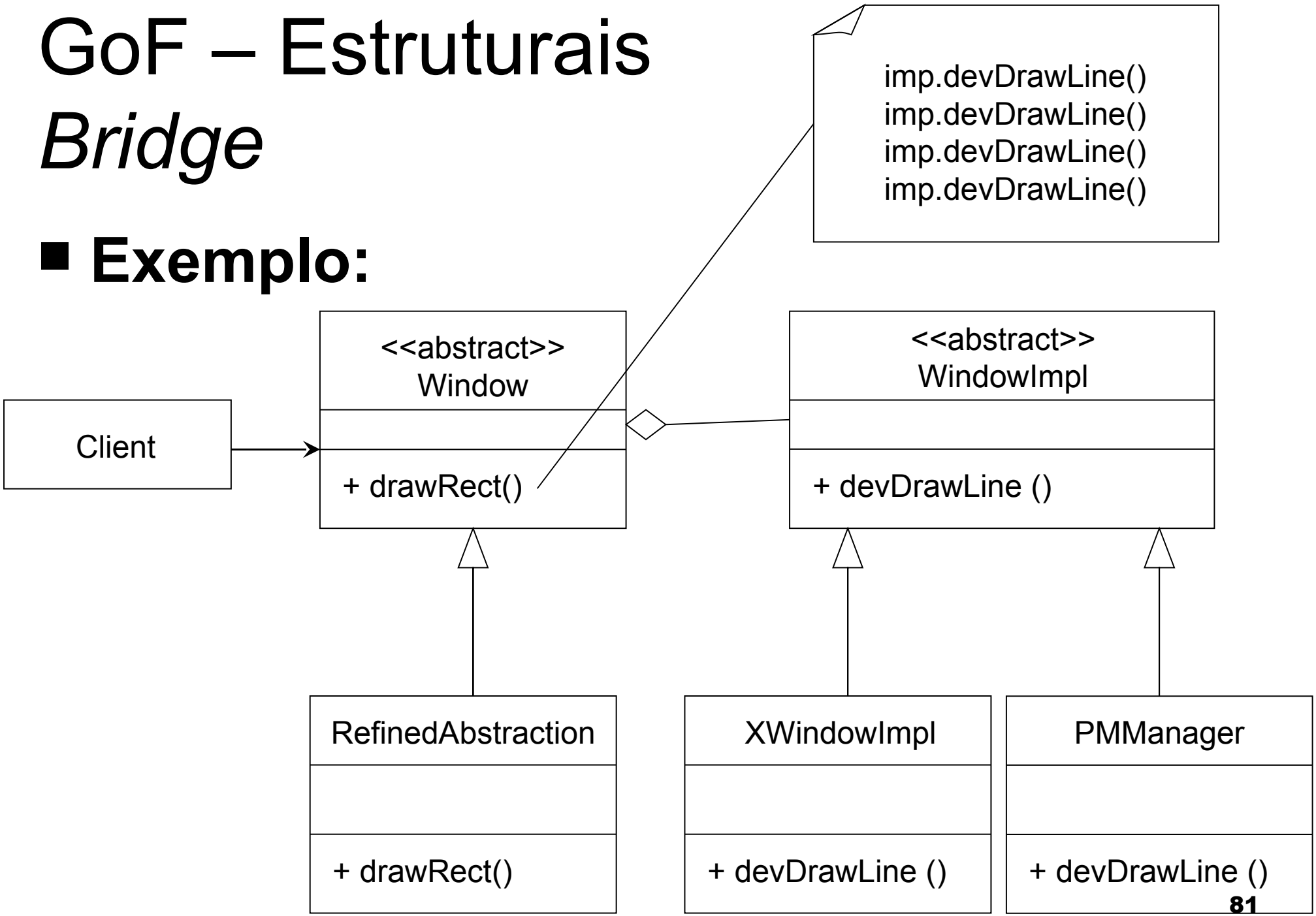
GoF – Estruturais
Bridge

- **Estrutura:**



GoF – Estruturais
Bridge

- **Exemplo:**



GoF – Estruturais
Bridge

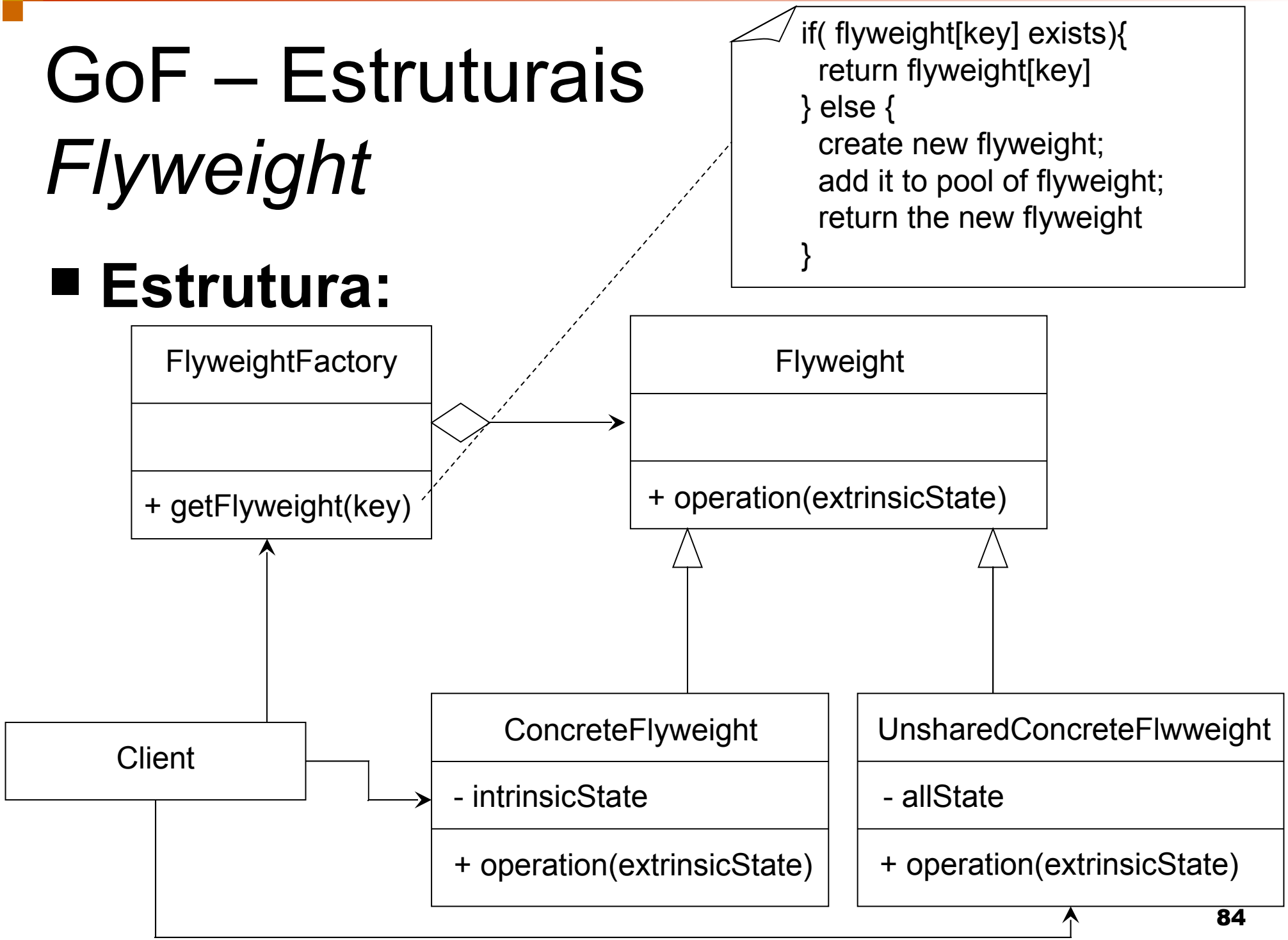
- **Use quando:**
  - Deseja evitar acoplamento permanente entre abstração e sua implementação
  - Abstração e implementação devem ser extensíveis
  - Mudanças na implementação não devem ter impactos nos clientes que usam a abstração

GoF – Estruturais
Flyweight

- **Intenção:** Usar compartilhamento para suportar eficientemente grandes quantidades de objetos com granularidade fina

GoF – Estruturais
Flyweight

- **Estrutura:**



GoF – Estruturais
Flyweight

- **Exemplo:**

```
class FlyweightFactory{
  private Flyweight pool[];
  public Flyweight getFlyweight(int key)
{
    if( pool[ key ] != null ){
      pool[key] =
        new ConcreteFlyweight();
    }
    return pool[key] ;
  }
}
```

GoF – Estruturais
Flyweight

- **Exemplo:**

```
class GoFTest{
   ...
   private fc FlyweightFactory;
   fc = new FlyweightFactory();
   Flyweight f =
     fc.getFlyweight( Flyweight.A );
   f.operation( newState ) ;
   f.run();
   ...
}
```

GoF – Estruturais
Flyweight

- **Use quando todos estes forem verdade:**
  - Uma aplicação usa um grande número de objetos
  - Custo de armazenagem é alto (muitos objetos)
  - Boa parte do estado do objeto pode ser extrínseca
  - Muitos grupos de objetos podem ser trocados por relativamente poucos objetos quando a parte extrínseca é removida
  - A aplicação não depende da identidade dos objetos, uma vez que serão compartilhados

Exercício

- Continuar projeto do sistema de fiscalização de velocidade
  - Incorporando padrões estruturais
  - Descrevendo métodos e atributos principais

GoF – Comportamentais (Capítulo 5)

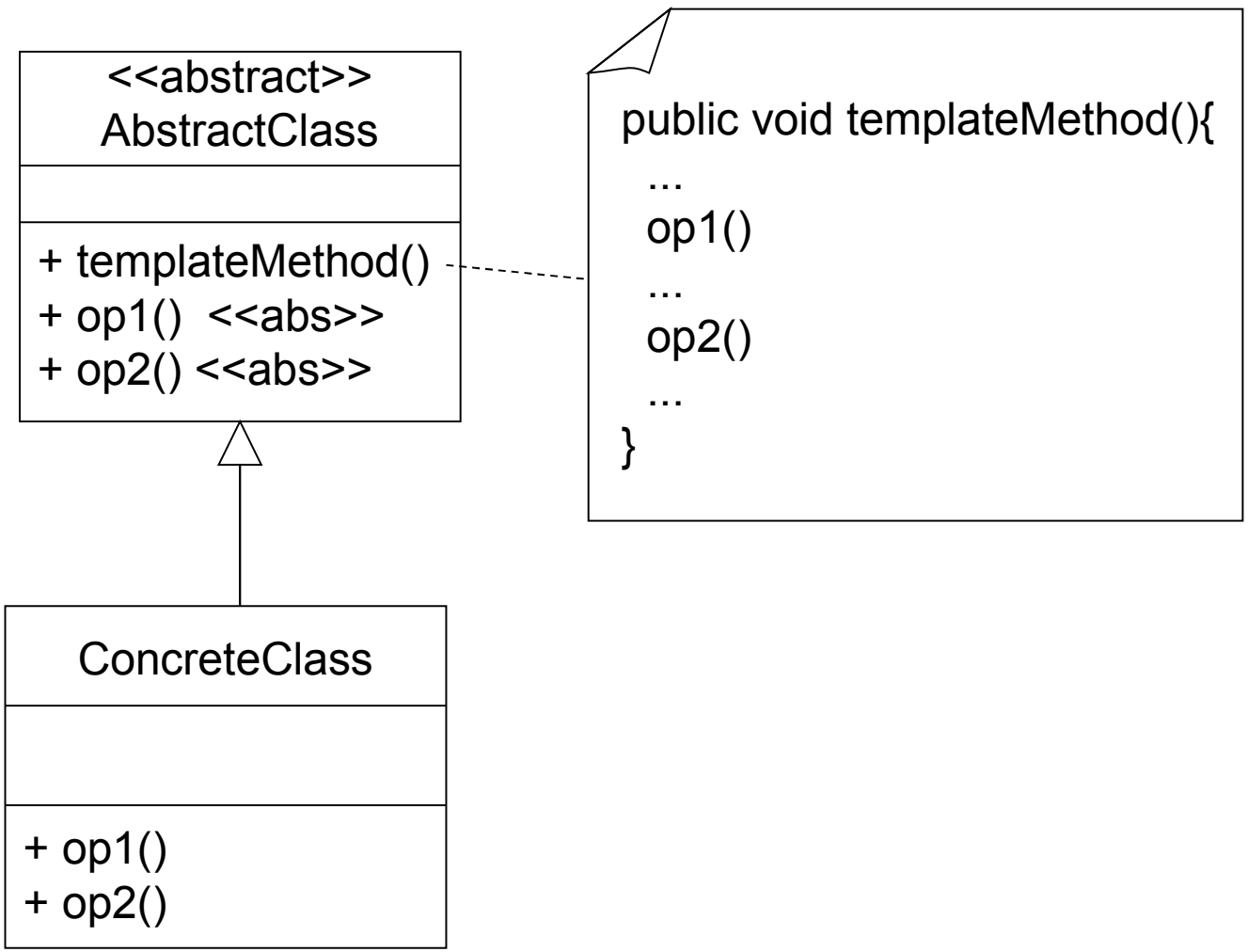
- *Template method*
- *Interpreter*
- *Mediator*
- *Chain of responsibility*
- *Observer*
- *State*
- *Strategy*
- *Command*
- *Memento*
- *Iterator*
- *Visitor*

GoF – Comportamentais
Template Method

- **Intenção:**
  - Definir o esqueleto de um algoritmo postergando alguns passos para as subclasses
  - As subclasses podem redefinir certos passos de um algoritmo sem mudar sua estrutura

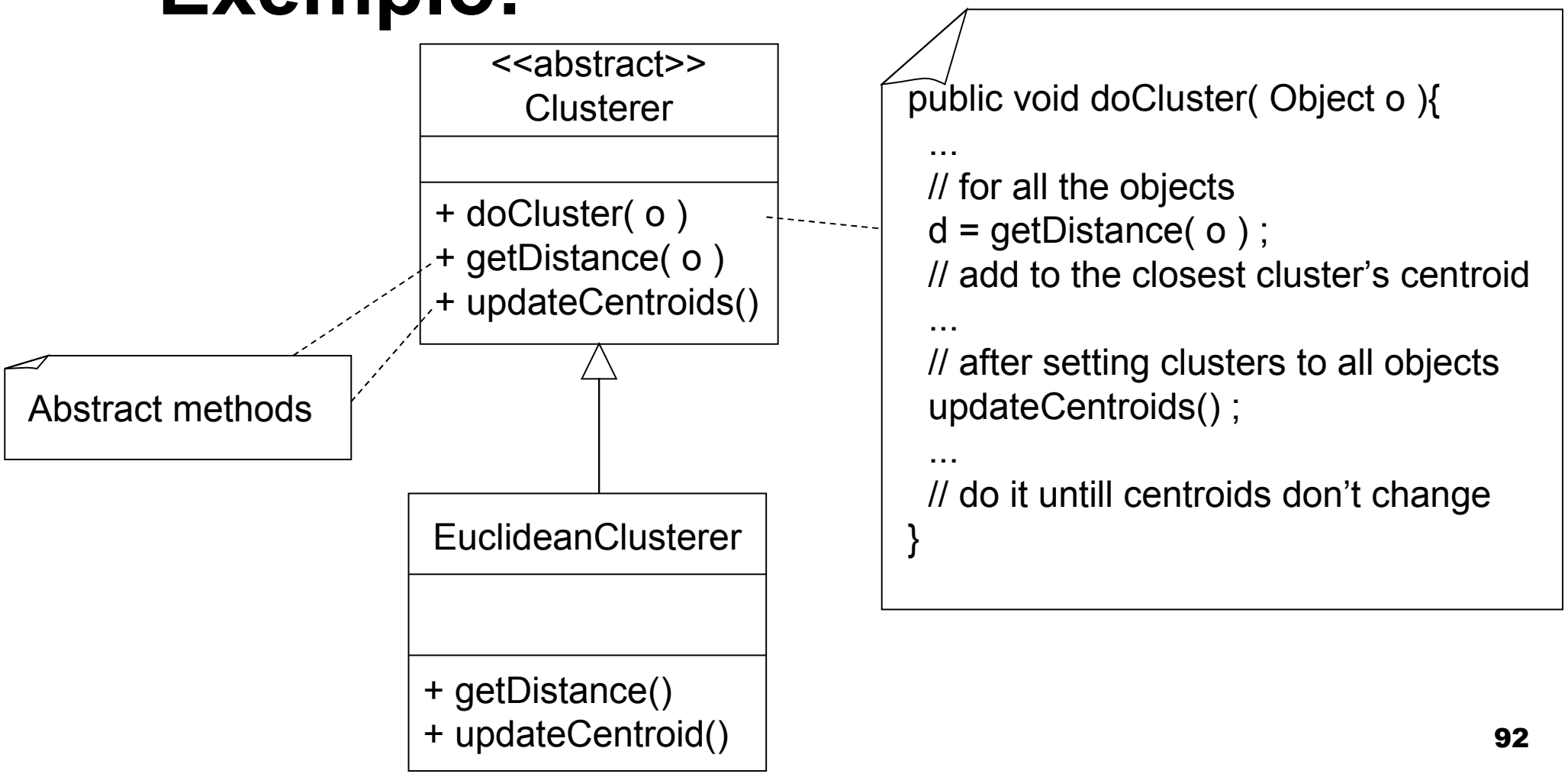
GoF – Comportamentais
Template Method

- **Estrutura:**



GoF – Comportamentais
Template Method

- **Exemplo:**



GoF – Comportamentais
Template Method

- **Use quando:**
  - Deseja implementar partes invariantes de um algoritmo uma vez e deixar que subclasses implementem o comportamento que pode variar
  - Um comportamento comum de subclasses pode ser divido e colocado em uma classe comum para evitar duplicação de código
  - Desejar controlar extensão de subclasses

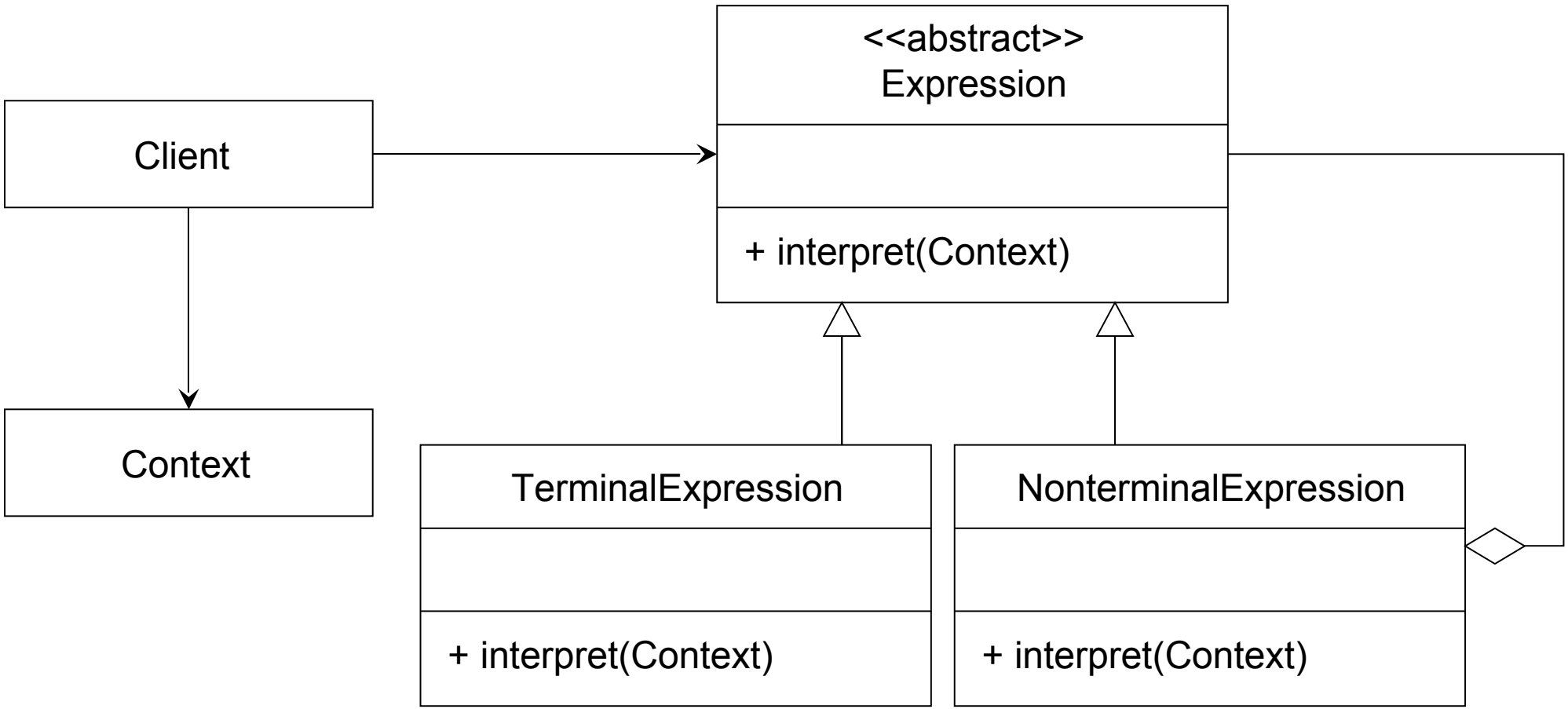
GoF – Comportamentais
Interpreter

- **Intenção:** Dada uma linguagem, definir uma representação para sua gramática juntamente com um interpretador que usa a representação para interpretar sentenças na linguagem

GoF – Comportamentais

Interpreter

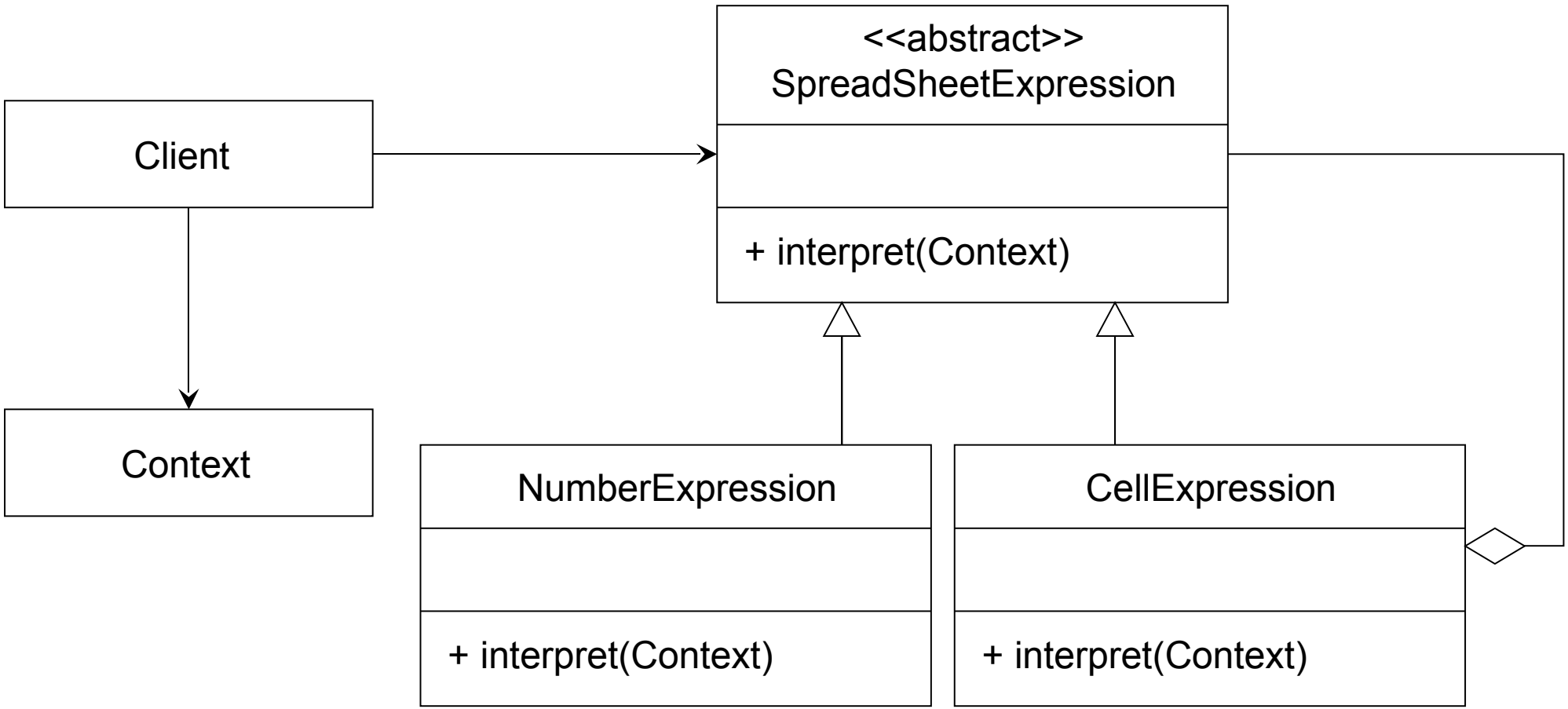
- **Estrutura:**



GoF – Comportamentais

Interpreter

- **Exemplo:**



GoF – Comportamentais
Interpreter

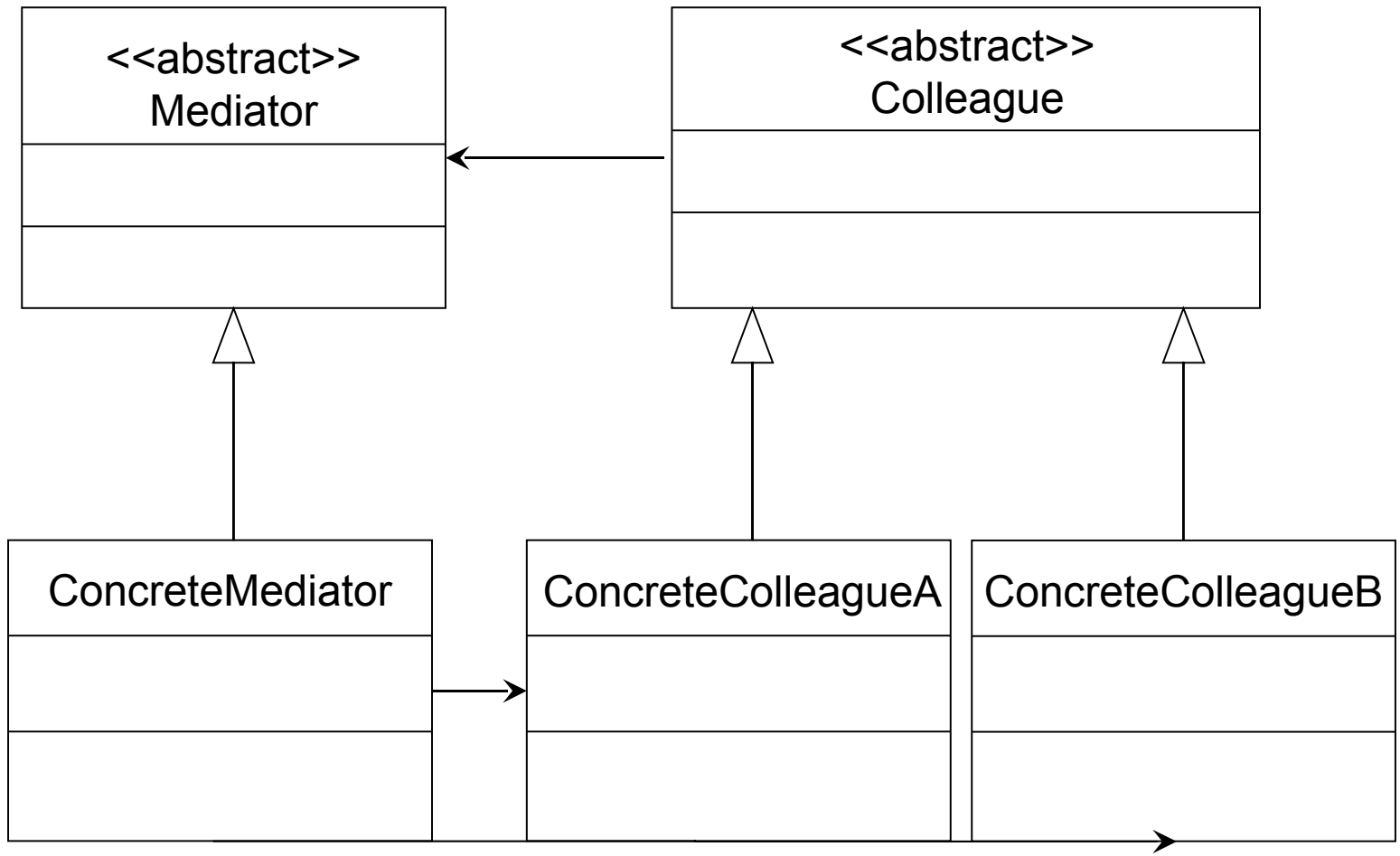
- **Use quando:**
  - Há uma gramática para interpretar e você pode representar sentenças dessa linguagem como árvores abstratas de sintaxe
  - Funciona melhor quando
    - A gramática é simples
    - Eficiência não é um ponto crítico

GoF – Comportamentais
Mediator

- **Intenção:**
  - Definir um objeto que encapsula a forma como um conjunto de objetos interage
  - Promove o baixo acoplamento evitando que objetos façam referência a outros explicitamente

GoF – Comportamentais
Mediator

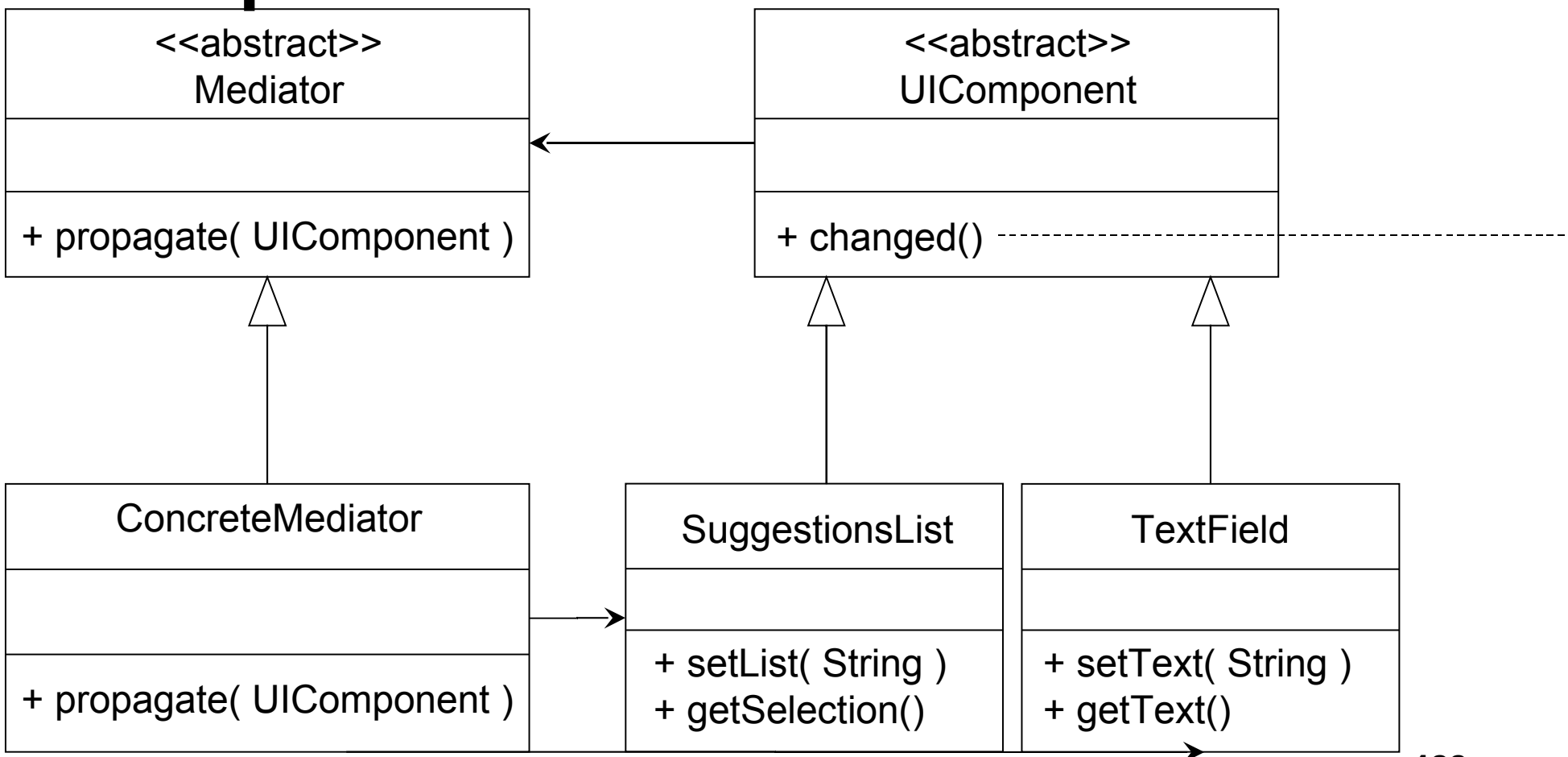
- **Estrutura:**



GoF – Comportamentais

Mediator

- **Exemplo:**



GoF – Comportamentais
Mediator

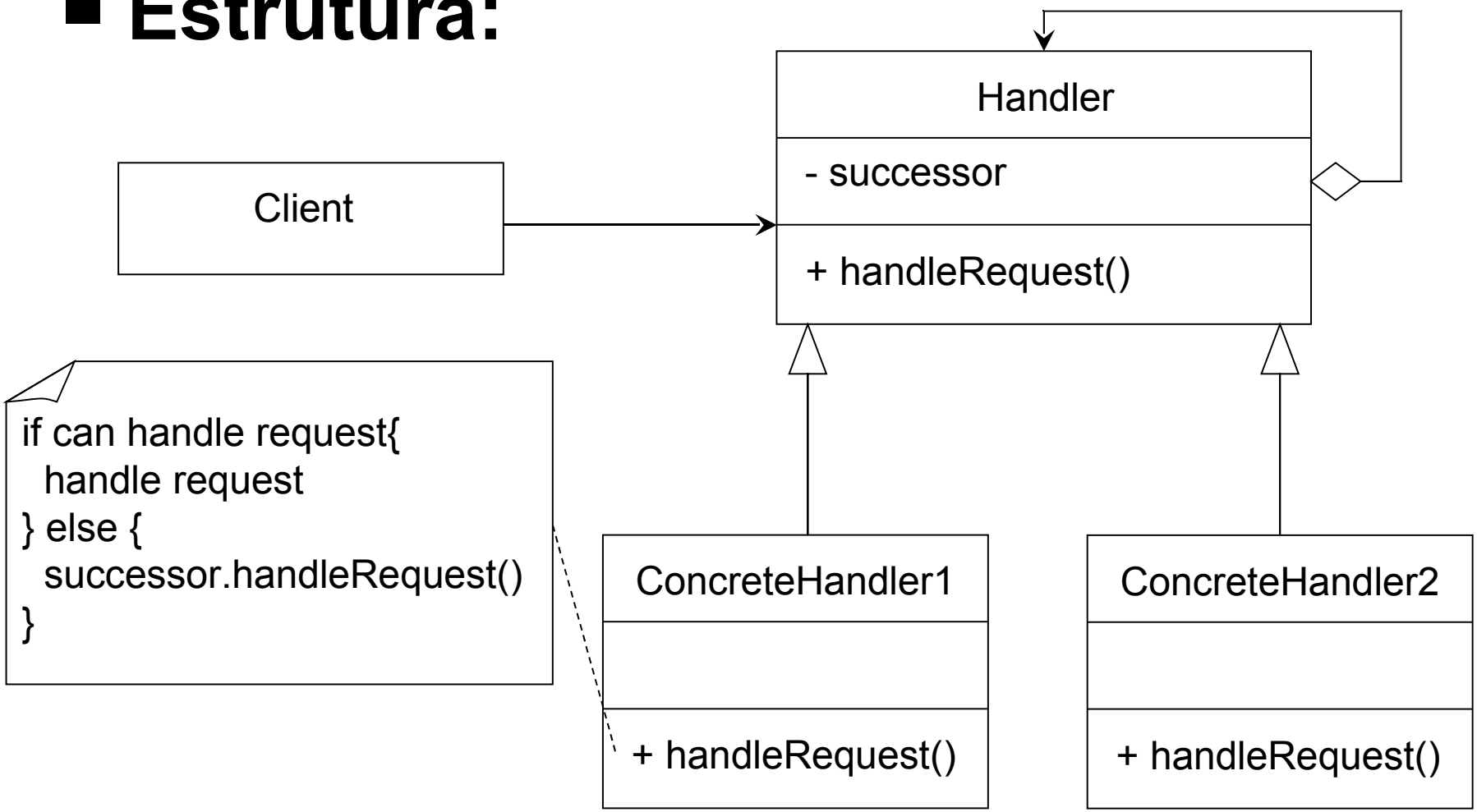
- **Use quando:**
  - Objetos se comunicam de maneira bem definida, mas complexa
  - Um objeto tem reúso restrito pois se comunica e referencia muitos objetos
  - Um comportamento que é distribuído entre várias classes deveria ser customizável sem utilizar muita herança

GoF – Comportamentais
Chain of Responsibility

- **Intenção:**
  - Evitar acoplamento entre remetente e destinatário de um pedido, dando a mais de um objeto a chance de responder um pedido
  - Encadeia objetos e passa *request* até que um deles responda

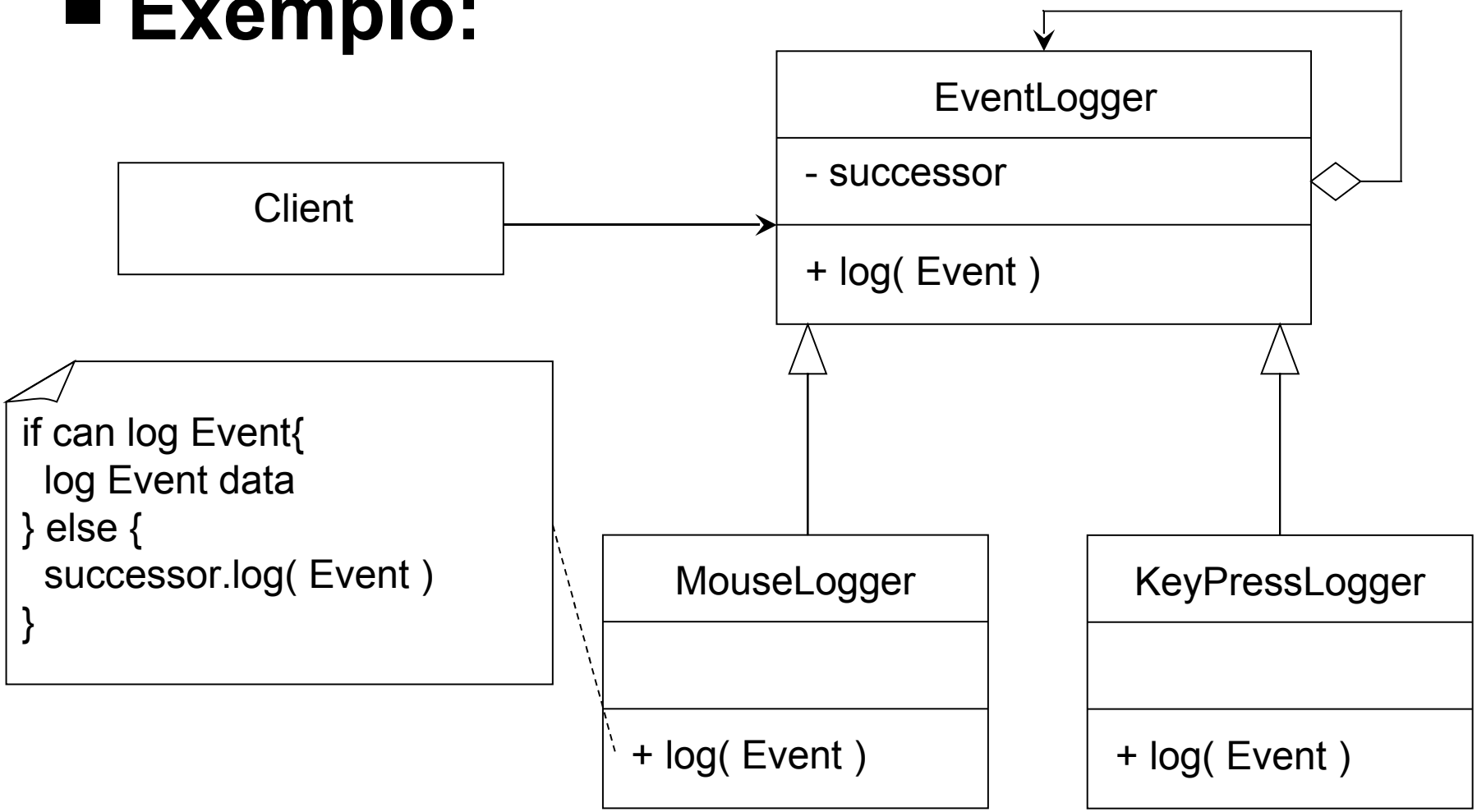
GoF – Comportamentais
Chain of Responsibility

- **Estrutura:**



GoF – Comportamentais
Chain of Responsibility

- **Exemplo:**



GoF – Comportamentais
Chain of Responsibility

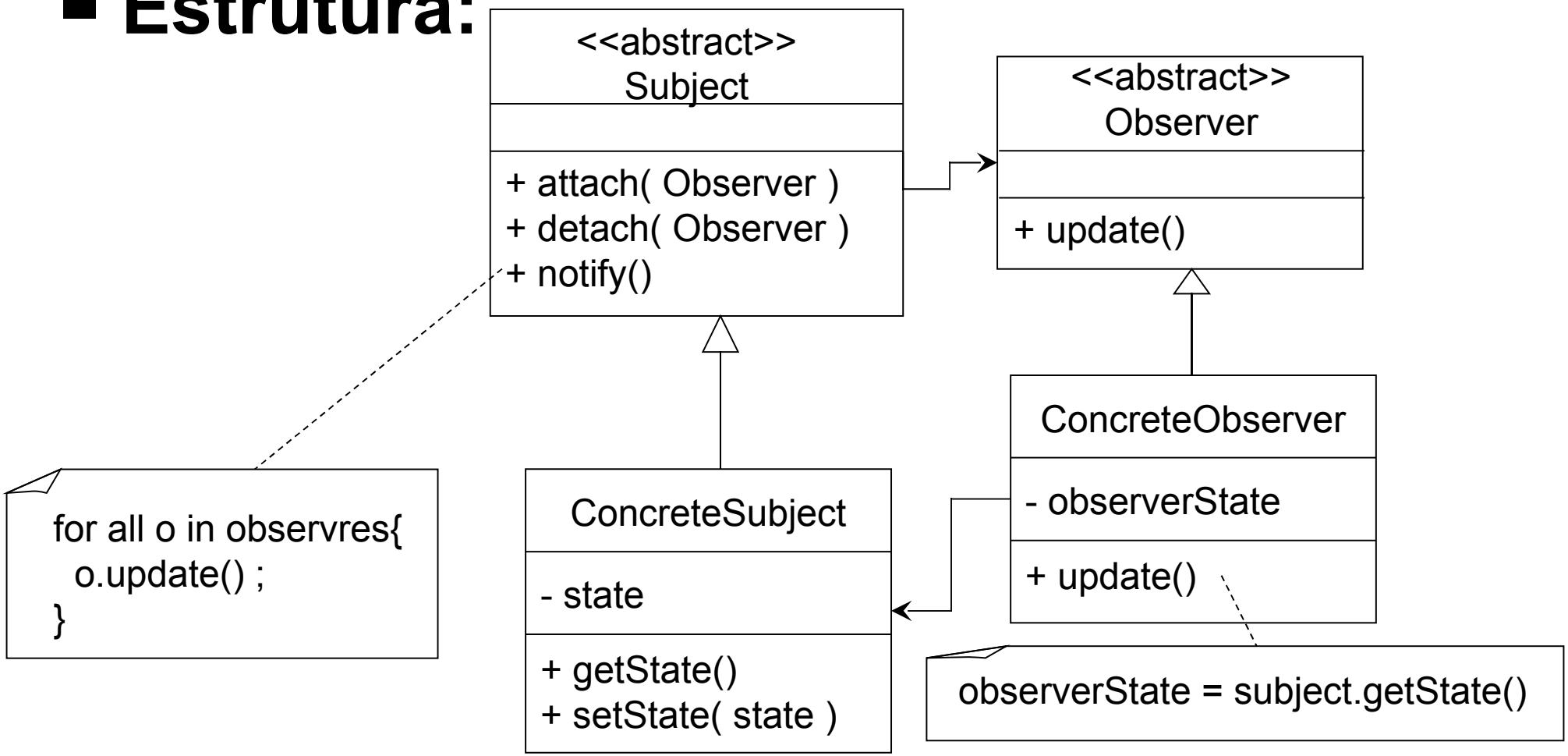
- **Use quando:**
  - Mais de um objeto pode manipular uma requisição e o *handler* não é conhecido de antemão. O *handler* deveria ser associado automaticamente
  - Você quer enviar uma requisição para vários objetos sem especificar o destinatário explicitamente
  - O conjunto de objetos que pode manipular uma requisição pode ser especificado dinamicamente

GoF – Comportamentais
Observer

- **Intenção:** Define uma interdependência **1 para n** entre objetos, de forma que quando um objeto muda de estado, todos os seus dependentes são notificados e atualizados

GoF – Comportamentais
Observer

- **Estrutura:**



GoF – Comportamentais
Observer

- **Exemplo:**

```
class ConcreteSubject extends Subject{
  private List<Observer> observers ; //N
  private State state ;
  ...
  public void attach( Observer o ){
    o.setSubject( this ) ;
    observers.add( o ) ;
  }
  ... (continua)
```

GoF – Comportamentais

Observer

- **Exemplo:**

```
public void detach( Observer o ){
  o.setSubject( null ) ;
  observers.remove( o ) ;
}
public void notify(){
  // i is iterator of observers list
  while(i.hasNext()){
    Observer o = i.next() ;
    o.update() ;
  }
}
}
```

GoF – Comportamentais
Observer

- **Exemplo:**

```
class ConcreteObserver extends Observer{
  private Subject subject ; //1
  private State state ;
  ...
  public void setSubject( Subject s ){
    subject = s ;
  }
  public void update(){
    state = subject.getState() ;
  }
}
```

GoF – Comportamentais
Observer

- **Exemplo:**

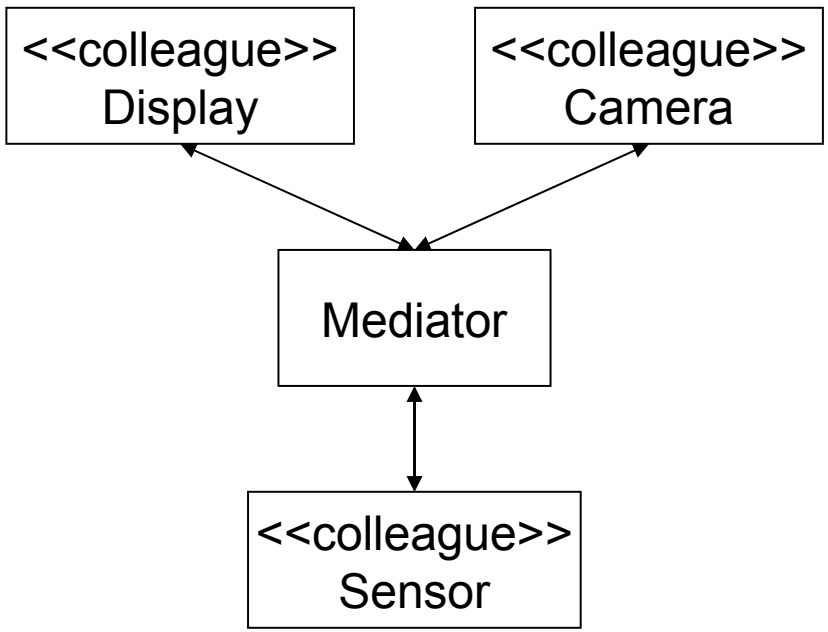
```
class GoFTest{
   ...
   public static void main( String args[] ){
      // 1 'source'
      Subject s = new ConcreteSubject() ;
      // N dependents
      Observer o1 = new ConcreteObserver() ;
      s.attach( o1 ) ;
      Observer o2 = new ConcreteObserver() ;
      s.attach( o2 ) ;
      ...
   }
}
```

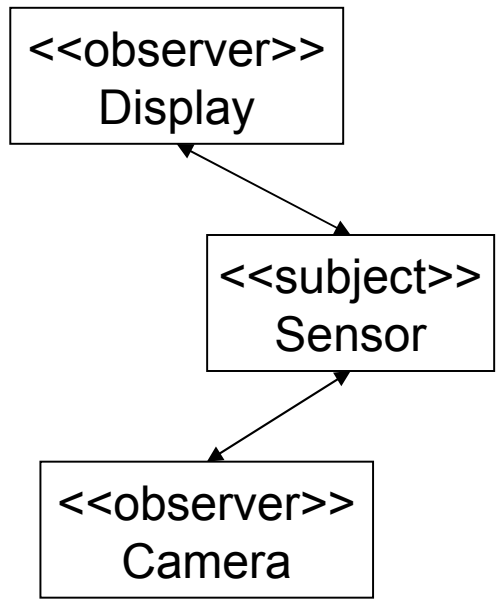
GoF – Comportamentais
Observer

- **Use quando:**
  - Uma abstração tiver dois aspectos, um dependente do outro. Encapsular esses aspectos em objetos separados possibilita reusá-los independentemente
  - Uma mudança **1:n** ocorre e você não sabe quantos objetos precisam ser alterados
  - Um objeto precisa notificar outros objetos sem fazer suposições sobre quem eles são

GoF – Comportamentais
Observer

- Considerando o sensor de um radar de velocidade, o que é mais adequado Mediator ou Observer?



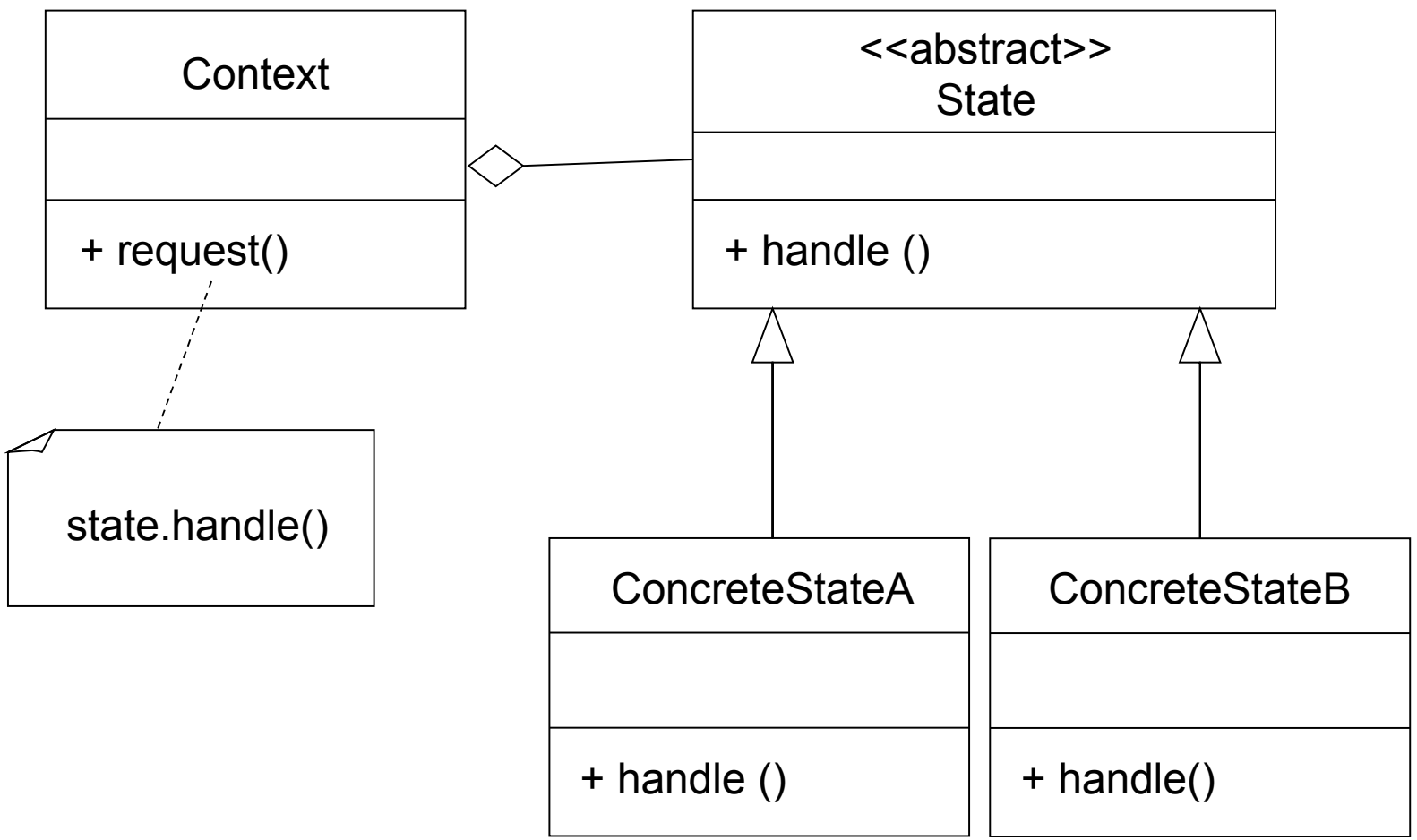


GoF – Comportamentais
State

- **Intenção:** Permite a um objeto alterar seu comportamento quando seu estado interno muda. O objeto parecerá ter "mudado de classe"

GoF – Comportamentais
State

- **Estrutura:**



GoF – Comportamentais

State

- **Exemplo:**

```
class Sensor{
   private State state ;
   public setState( State s ){
      state = s ;
   }
   public int trackSpeed(){
      return state.handle( this ) ;
   }
}
```

GoF – Comportamentais
State

- **Exemplo:**

```
abstract class State{
   abstract void handle( Sensor s ) ;
}
class CarPassing extends State{
   public int handle( Sensor s ){
      int speed = s.getSpeed() ;
      s.setState( new NoCarPassing() );
      return speed ;
   }
}
```

GoF – Comportamentais
State

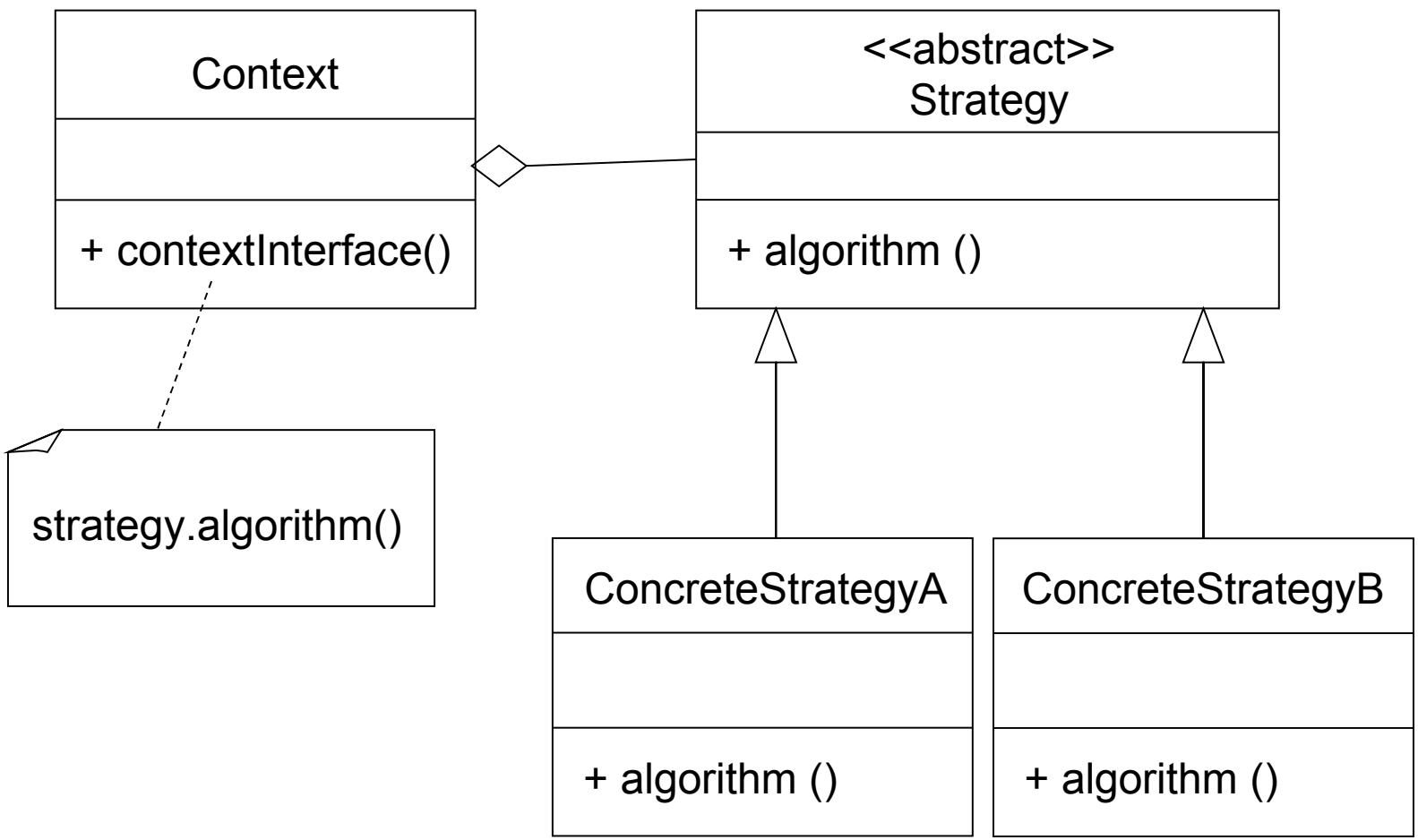
- **Use quando:**
  - O comportamento de um objeto depende do seu estado ele deve mudá-lo em tempo de execução
  - Operações tem vários fluxos condicionais que dependem do estado do objeto. *State* coloca cada um desses fluxos em uma classe separada

GoF – Comportamentais
Strategy

- **Intenção:**
  - Define uma família de algoritmos, encapsula cada um deles e os torna intercambiáveis
  - *Strategy permite* que o algoritmo varie independentemente dos clientes que a utilizam

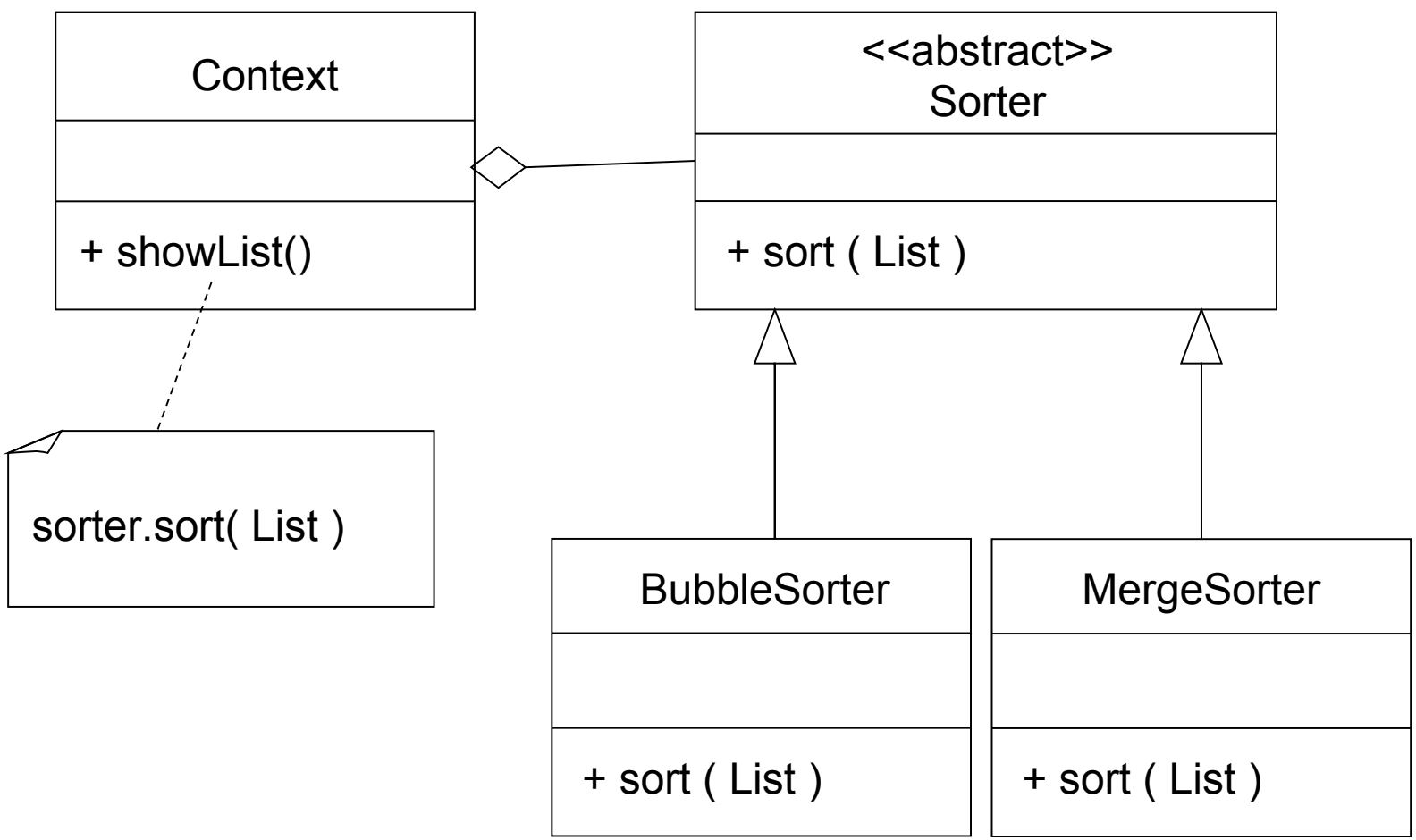
GoF – Comportamentais
Strategy

- **Estrutura:**



GoF – Comportamentais
Strategy

- **Exemplo:**



GoF – Comportamentais
Strategy

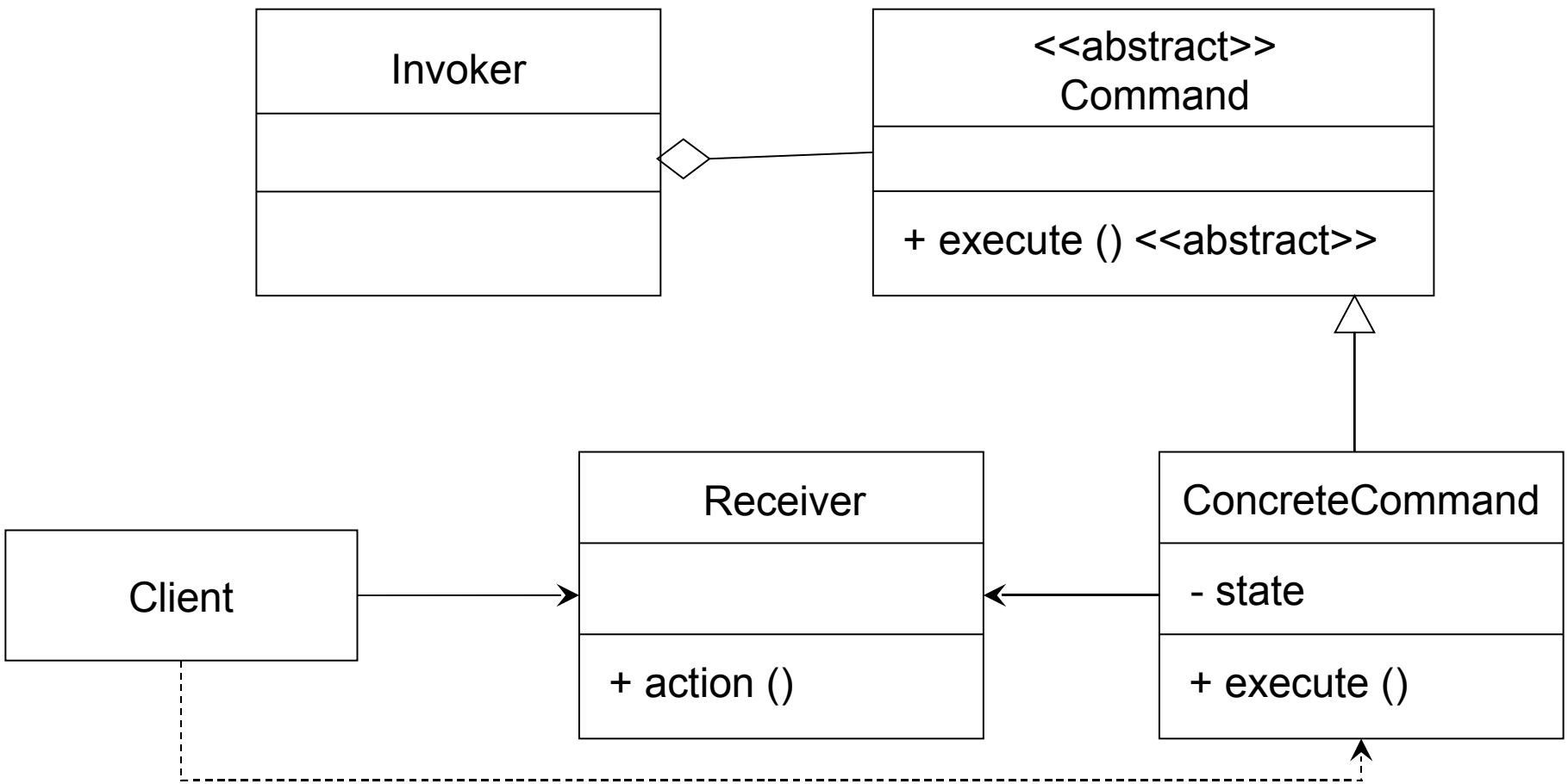
- **Use quando:**
  - Várias classes relacionadas diferem somente no comportamento
  - Precisa de variantes de um certo algoritmo
  - Um algoritmo usa dados que clientes não precisam ter conhecimento
  - Uma classe define muitos comportamentos que aparecem em vários fluxos condicionais

GoF – Comportamentais
Command

- **Intenção:**
  - Encapsular uma requisição em um objeto, permitindo
    - parametrizar clientes
    - enfileirar requisições
    - registrar requisições
    - suportar operações que podem ser desfeitas

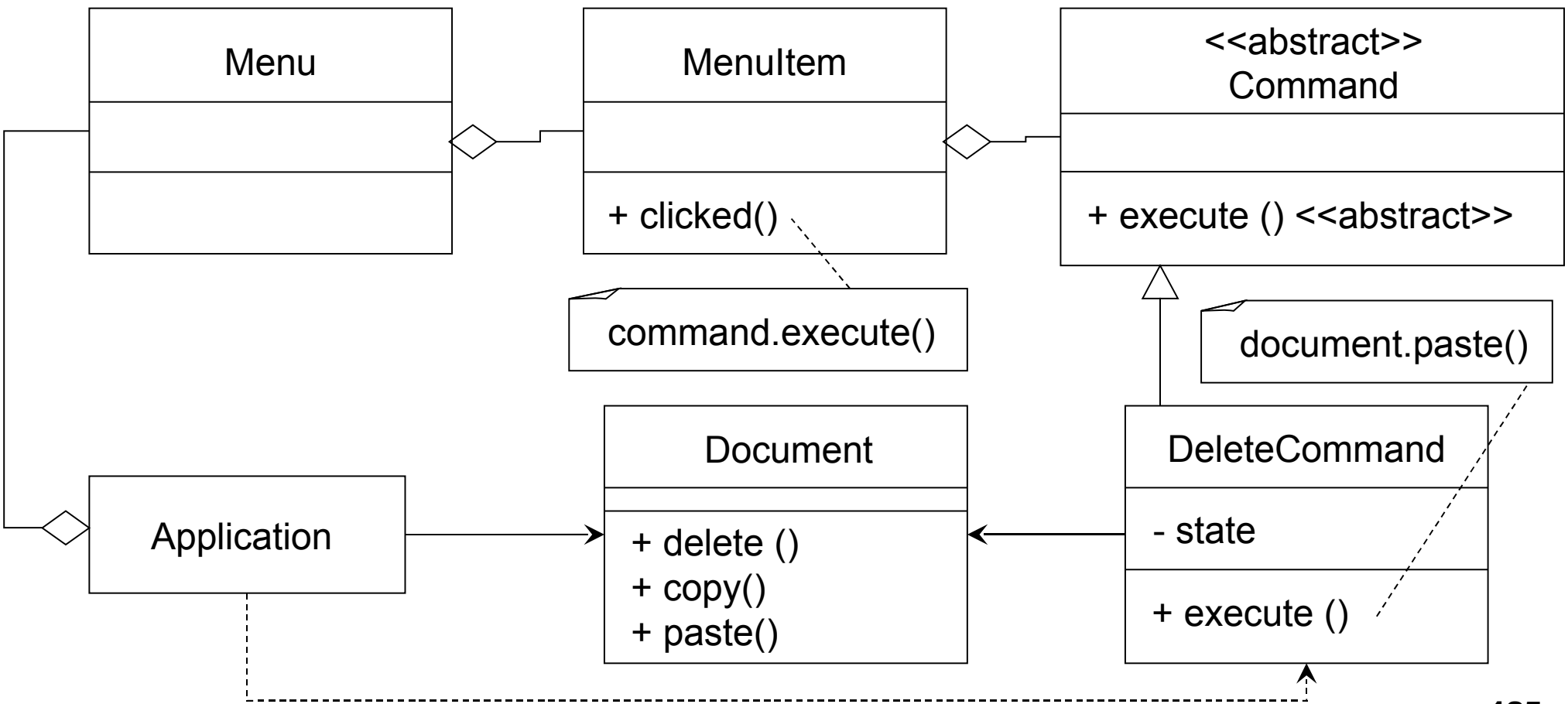
GoF – Comportamentais
Command

- **Estrutura:**



GoF – Comportamentais
Command

- **Exemplo:**



GoF – Comportamentais
Command

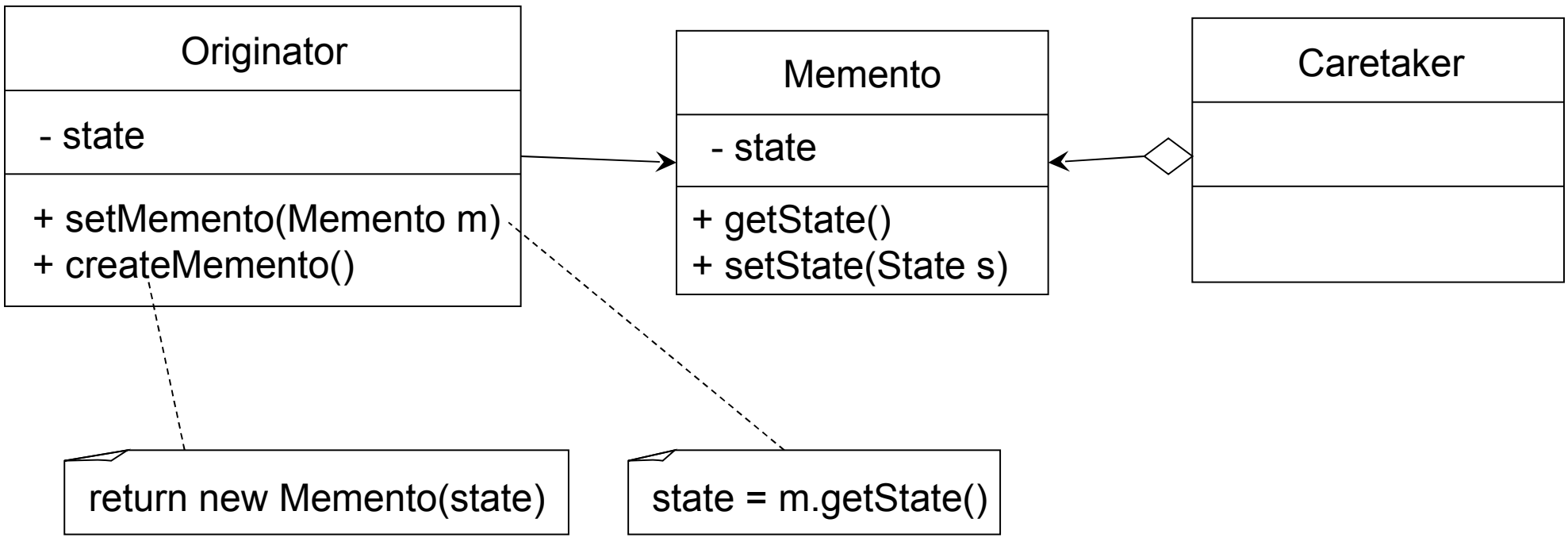
- **Use quando:**
  - Desejar parametrizar objetos com base na ação a ser executada
  - Desejar especificar filas e executar solicitações em tempos diferentes
  - Desejar implementar “desfazer”
  - Desejar implementar *logging* de ações para serem reaplicadas em sistemas em caso de crash
  - Desejar estruturar um sistema em operações em alto nível construídas com base em operações primitivas

GoF – Comportamentais
Memento

- **Intenção:** Sem violar encapsulamento, capturar e externalizar o estado interno de um objeto de maneira que o objeto possa retornar a esse estado mais tarde

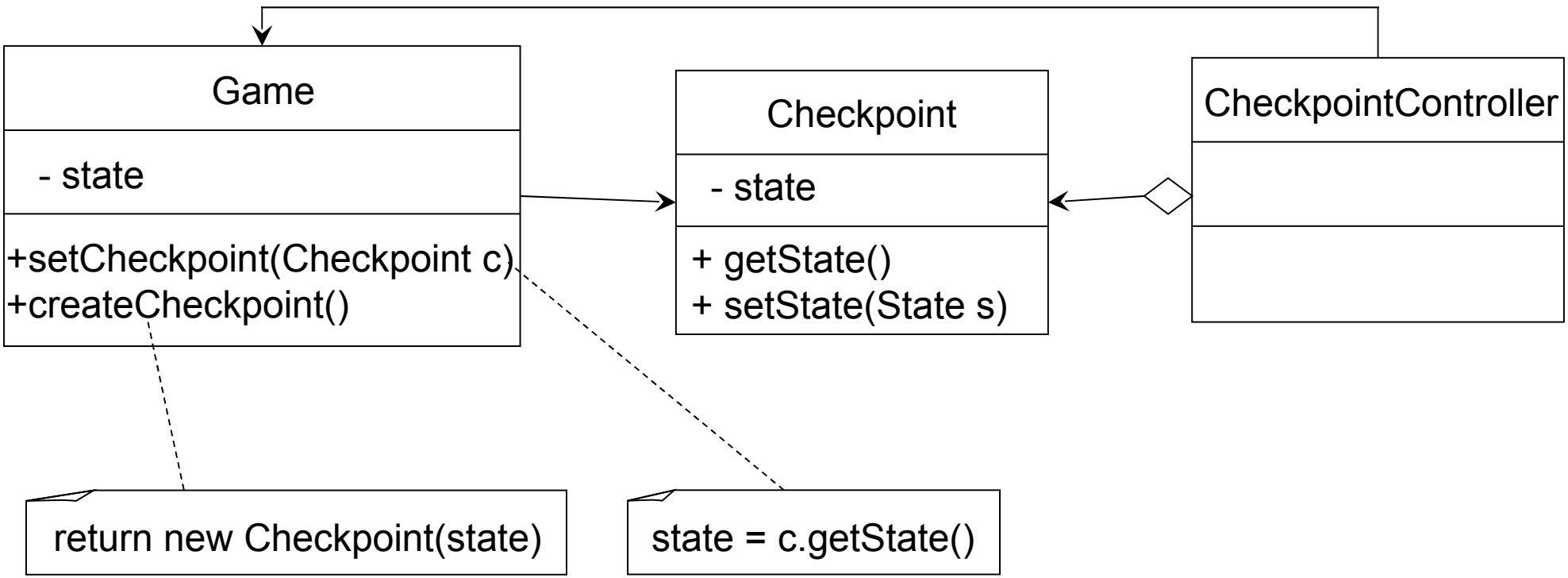
GoF – Comportamentais
Memento

- **Estrutura:**



GoF – Comportamentais
Memento

- **Exemplo:**



GoF – Comportamentais
Memento

- **Use quando:**
  - Um *snapshot* de alguma parte do objeto precisa ser salva para que seja restaurada no futuro; e
  - Uma interface direta para obter o estado expõe detalhes de implementação e quebraria o encapsulamento do objeto

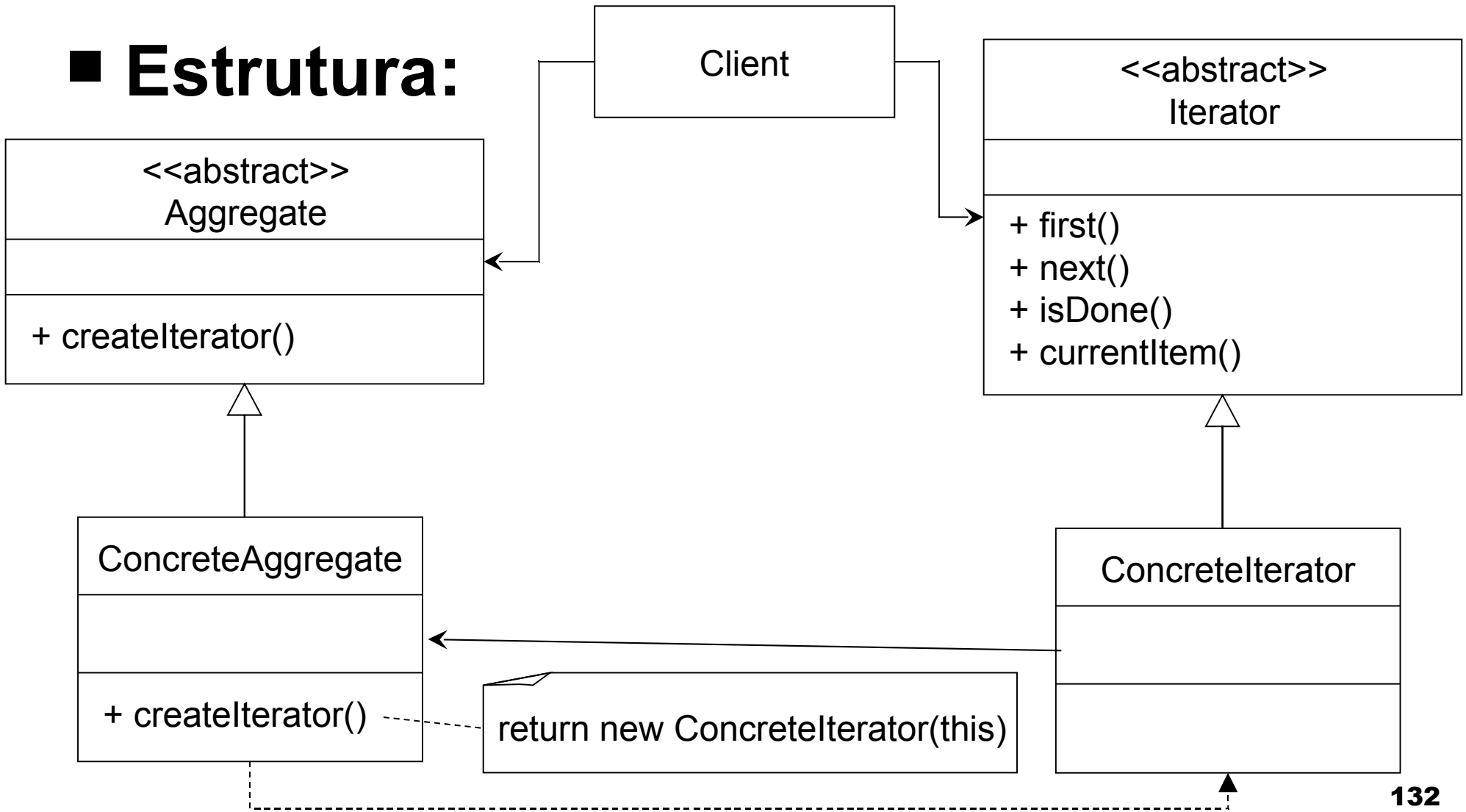
GoF – Comportamentais
Iterator

- **Intenção:** Fornecer um meio de acessar, sequencialmente, os elementos de uma agregação sem expor sua representação interna

GoF – Comportamentais

Iterator

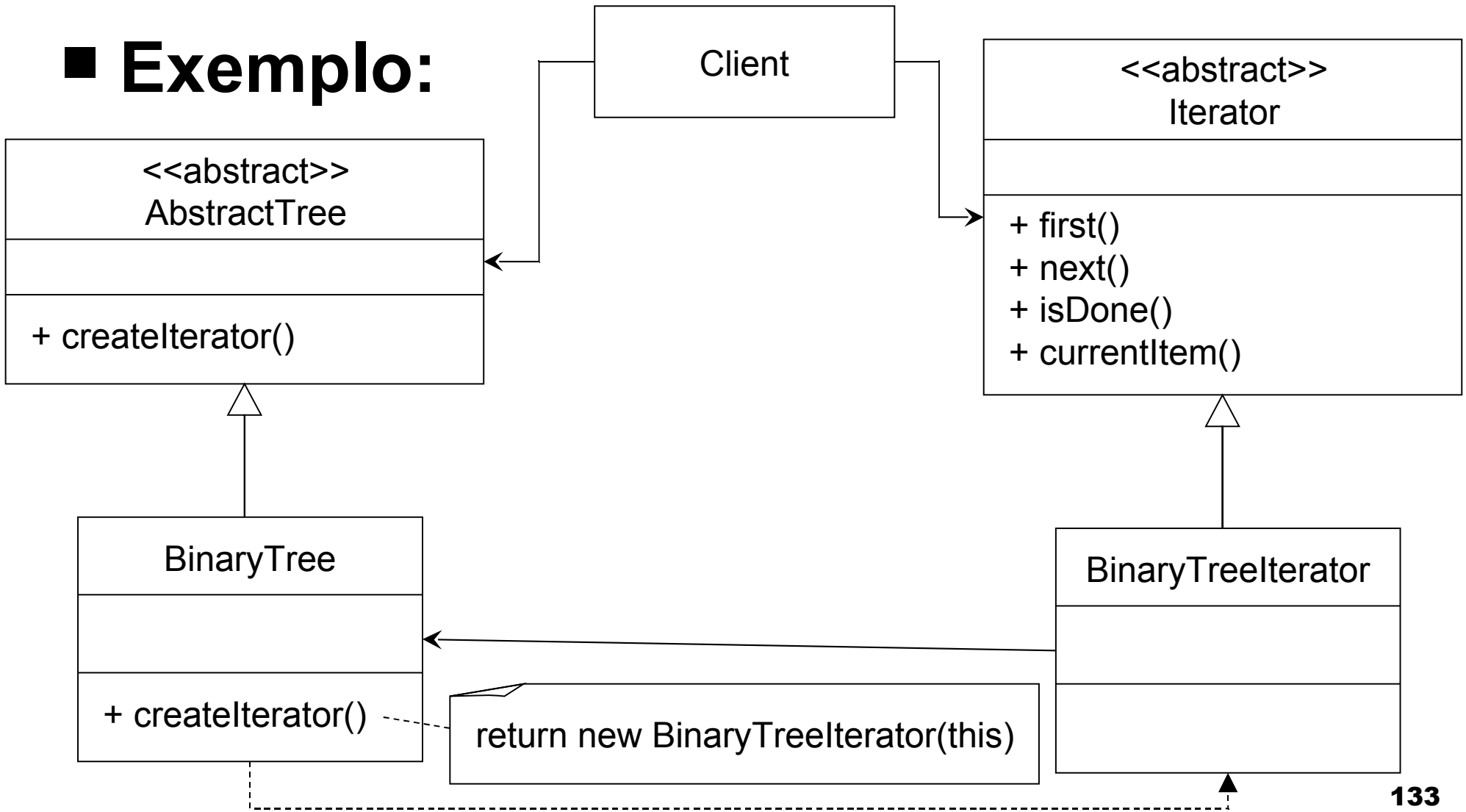
- **Estrutura:**



GoF – Comportamentais

Iterator

- **Exemplo:**



GoF – Comportamentais
Iterator

- **Use quando:**
  - Quiser acessar conteúdo de uma coleção de objetos sem expor sua representação interna
  - Quiser fornecer uma interface uniforme para navegar em diferentes estruturas de coleções de objetos (suportar iteração polimórfica)

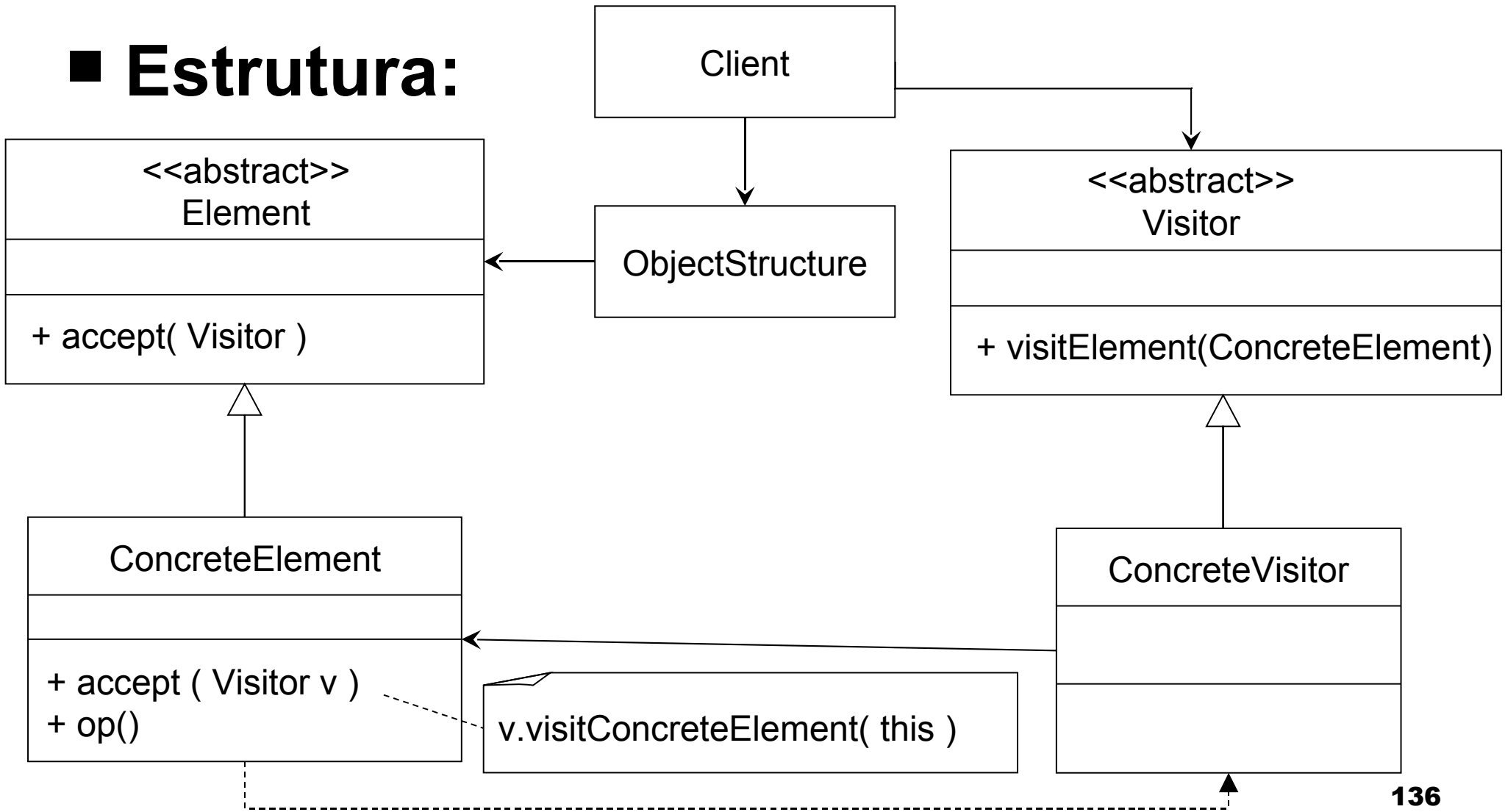
GoF – Comportamentais
Visitor

- **Intenção:**
  - Representar uma operação a ser executada nos elementos de uma estrutura de objetos.
  - *Visitor* permite definir uma nova operação sem mudar a classe dos elementos sobre os quais opera

GoF – Comportamentais

Visitor

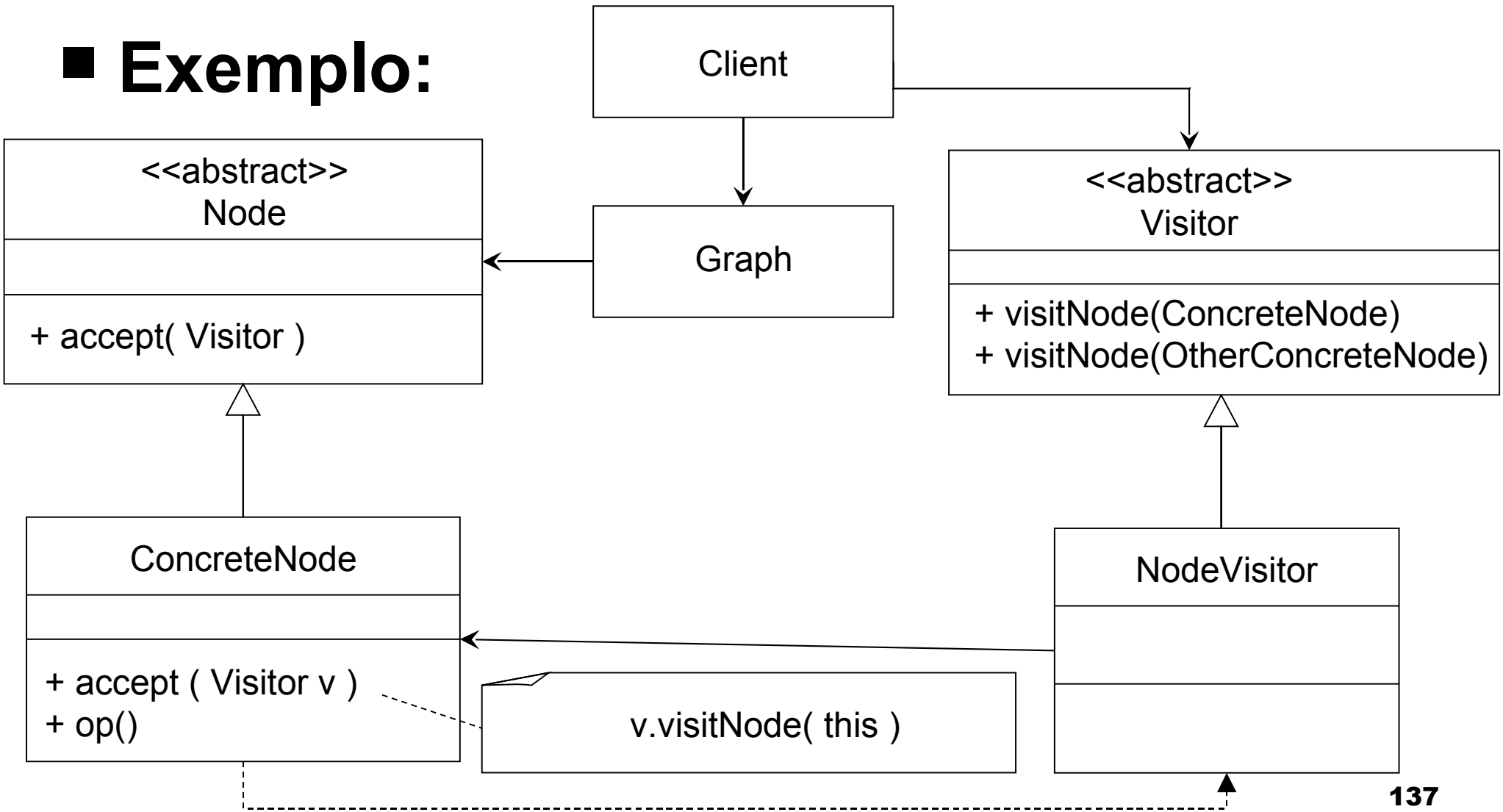
- **Estrutura:**



GoF – Comportamentais

Visitor

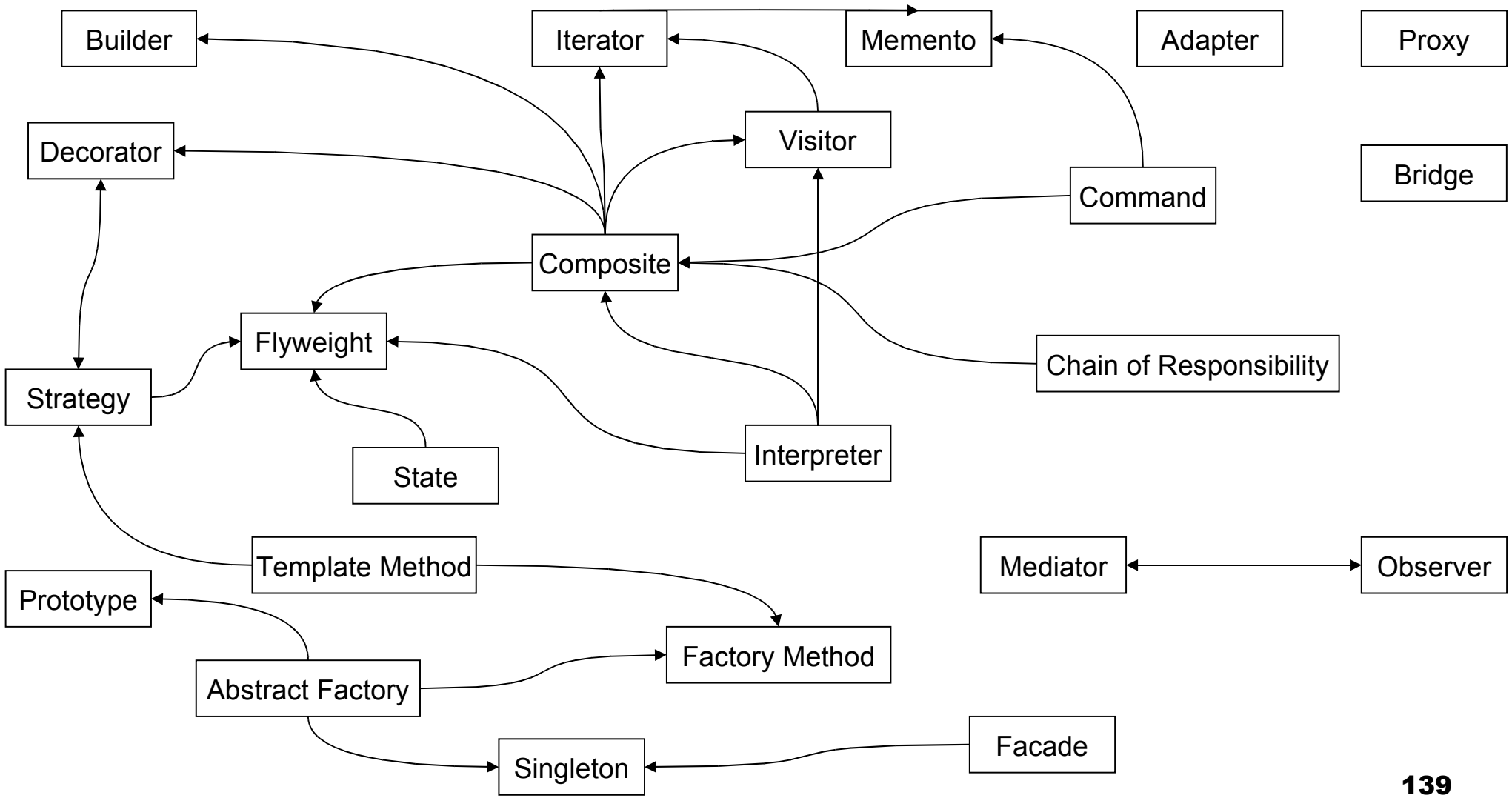
- **Exemplo:**



GoF – Comportamentais
Visitor

- **Use quando:**
  - A estrutura de um objeto contém muitas classes de objetos com interfaces diferentes e você deseja realizar operações nesses objetos
  - Operações diferentes e não relacionadas precisam ser aplicadas em uma estrutura de objetos e você não deseja “poluí-los” com essas operações
  - Classes definindo estruturas raramente mudam, mas comumente você deseja definir novas operações nessa estrutura

GoF
Relacionamento entre padrões



Exercício

- Finalizar projeto do sistema de fiscalização de velocidade para apresentação na próxima aula
  - 20min por grupo
  - 10min de discussões

Referências

- **Chidamber & Kemerer**
  A Metrics Suite for Object Oriented Design
- **Gamma et al.**
  Design Patterns: Elements of Reusable Object-Oriented Software (1994)
- **Larman**
  Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)