Behavior Patterns

Matheus Freitas e Leonardo Schroter - Sistemas de Informação - UFN

Introdução a categoria

- **Descrição:**
  - Os padrões comportamentais se concentram em algoritmos e atribuições de responsabilidades entre objetos.

  - Eles ajudam na comunicação eficiente entre objetos e na definição de como interagem entre si.

Padrão observer

- O padrão Observer define uma dependência um-para-muitos entre objetos, de modo que quando um objeto muda de estado, todos os seus dependentes são notificados e atualizados automaticamente.

- Intenção e Propósito:
  - Permite que um objeto notifique outros objetos sobre mudanças de estado sem acoplamento excessivo.

Observer - Problema que resolve

- **Descrição:**
  - Necessidade de notificar múltiplos objetos sobre mudanças de estado de maneira eficiente.
  - Evitar o acoplamento rígido entre o objeto que muda de estado e os objetos que precisam ser notificados.

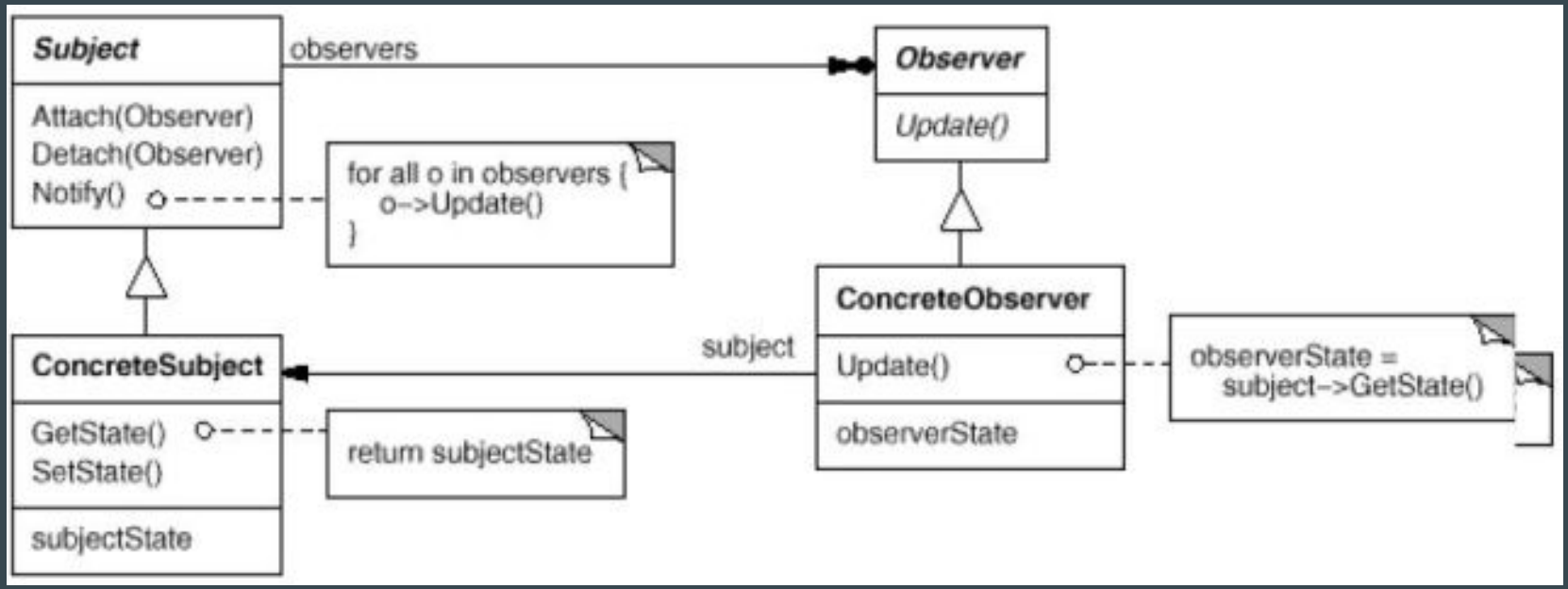
- **Exemplo de Contextualização:**
  - Em uma aplicação de notícias, quando uma nova notícia é publicada, todos os assinantes devem ser notificados.

Observer - Solução proposta

- **Descrição:**
  - Define um mecanismo para que os objetos "subject" mantenham uma lista de dependentes, chamados "observers".
  - Quando o estado do "subject" muda, ele notifica todos os "observers" registrados.

- **Diagrama de classes:**
  - Subject: Mantém uma lista de observers e notifica-os sobre mudanças de estado.
  - Observer: Define uma interface para atualização em resposta a mudanças no subject.
  - ConcreteSubject: Armazena o estado de interesse para ConcreteObservers.
  - ConcreteObserver: Implementa a interface de atualização para manter seu estado consistente com o subject.

Observer - Diagrama de classes



Observer - Quando Usar

Cenários Específicos:

- Sistemas de notificação
- Implementação de assinaturas/publicações

Condições Justificáveis:

- Necessidade de múltiplas partes reagindo a mudanças de estado
- Promoção de um acoplamento fraco entre objetos

Observer - Exemplo Prático

Exemplo de Implementação:

https://refactoring.guru/pt-br/design-patterns/observer/java/example

Observer - Vantagens e Desvantagens

Vantagens:

- Promove o baixo acoplamento
- Flexibilidade na adição/removal de observadores

Desvantagens:

- Potencial impacto de performance com muitos observadores
- Complexidade aumentada em sistemas grandes

Padrão Strategy

- **Descrição:**
  - O padrão Strategy define uma família de algoritmos, encapsula cada um deles e os torna intercambiáveis. Permite que o algoritmo varie independentemente dos clientes que o utilizam.

- **Intenção e Propósito:**
  - Permitir que diferentes algoritmos sejam selecionados em tempo de execução para realizar uma tarefa específica.

Strategy - Problemas que resolve

- **Descrição:**
  - Necessidade de alterar o comportamento de um algoritmo em tempo de execução sem modificar o código cliente.

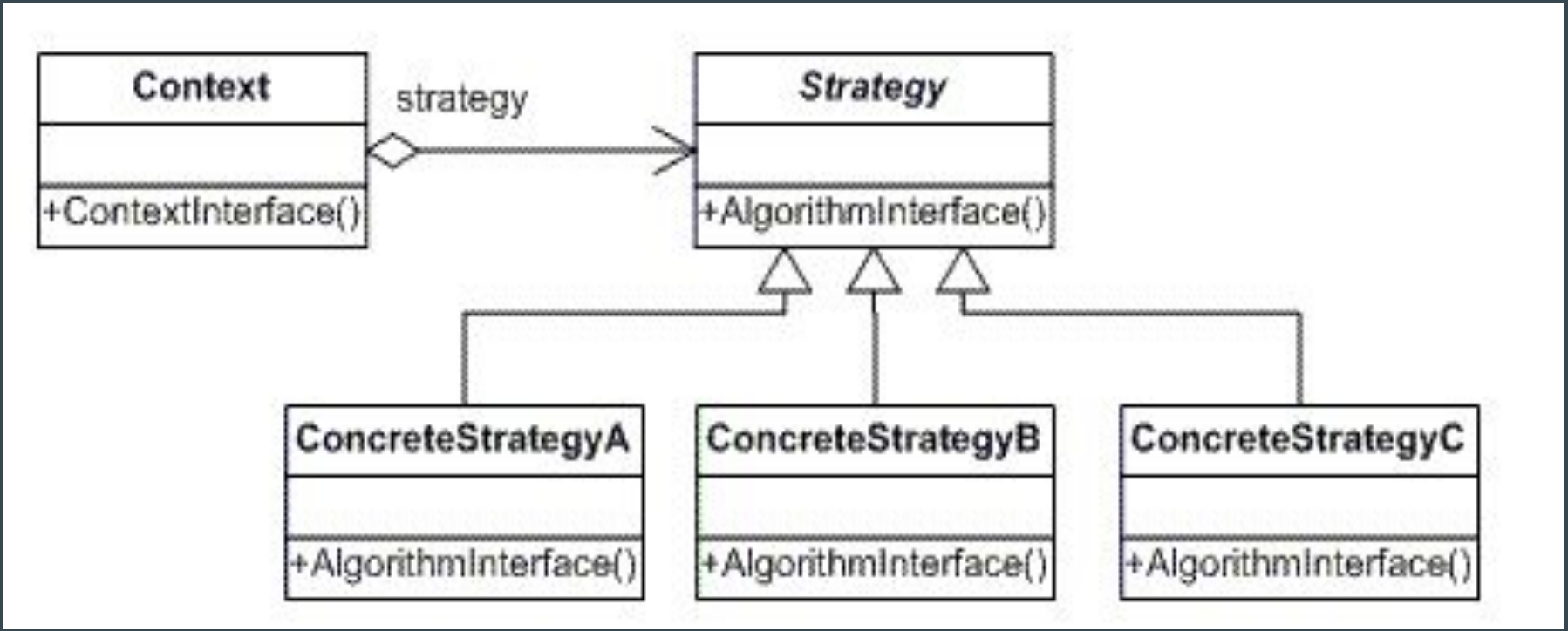
- **Exemplo de Contextualização:**
  - Em um editor de texto, diferentes algoritmos de ordenação podem ser aplicados a uma lista de palavras dependendo do critério escolhido pelo usuário (alfabética, por comprimento, etc.).

Strategy - Solução proposta

- **Descrição:**
  - Define uma interface comum para todos os algoritmos, permitindo que sejam substituíveis.
  - Cada algoritmo concreto implementa a interface de maneira diferente.

- **Diagrama de Classes:**
  - Context: Mantém uma referência a um objeto Strategy.
  - Strategy: Interface comum para todos os algoritmos.
  - ConcreteStrategy: Implementações específicas dos algoritmos.

Strategy - Diagrama de classes



Strategy - Quando Usar

Cenários Específicos:

- Algoritmos intercambiáveis
- Necessidade de variar o comportamento de um objeto

Condições Justificáveis:

- Implementação de famílias de algoritmos
- Separação de comportamento e algoritmo em classes distintas

Strategy - Exemplo Prático

https://refactoring.guru/pt-br/design-patterns/strategy/java/example

Strategy - Vantagens e Desvantagens

Vantagens:

- Facilita a intercambialidade de algoritmos
- Promove a extensibilidade

Desvantagens:

- Pode aumentar o número de objetos/classes
- Complexidade de manutenção com muitas estratégias

Padrão Mediator

- **Descrição:**
  - O padrão Mediator define um objeto que encapsula a forma como um conjunto de objetos interage. Promove o desacoplamento ao evitar que os objetos se refiram uns aos outros explicitamente.

- **Intenção e Propósito:**
  - Facilitar a comunicação entre objetos, evitando referências diretas entre eles e promovendo um design mais flexível.

Mediator - Problemas que resolve

- **Descrição:**
  - Complexidade e acoplamento excessivo devido à comunicação direta entre muitos objetos.

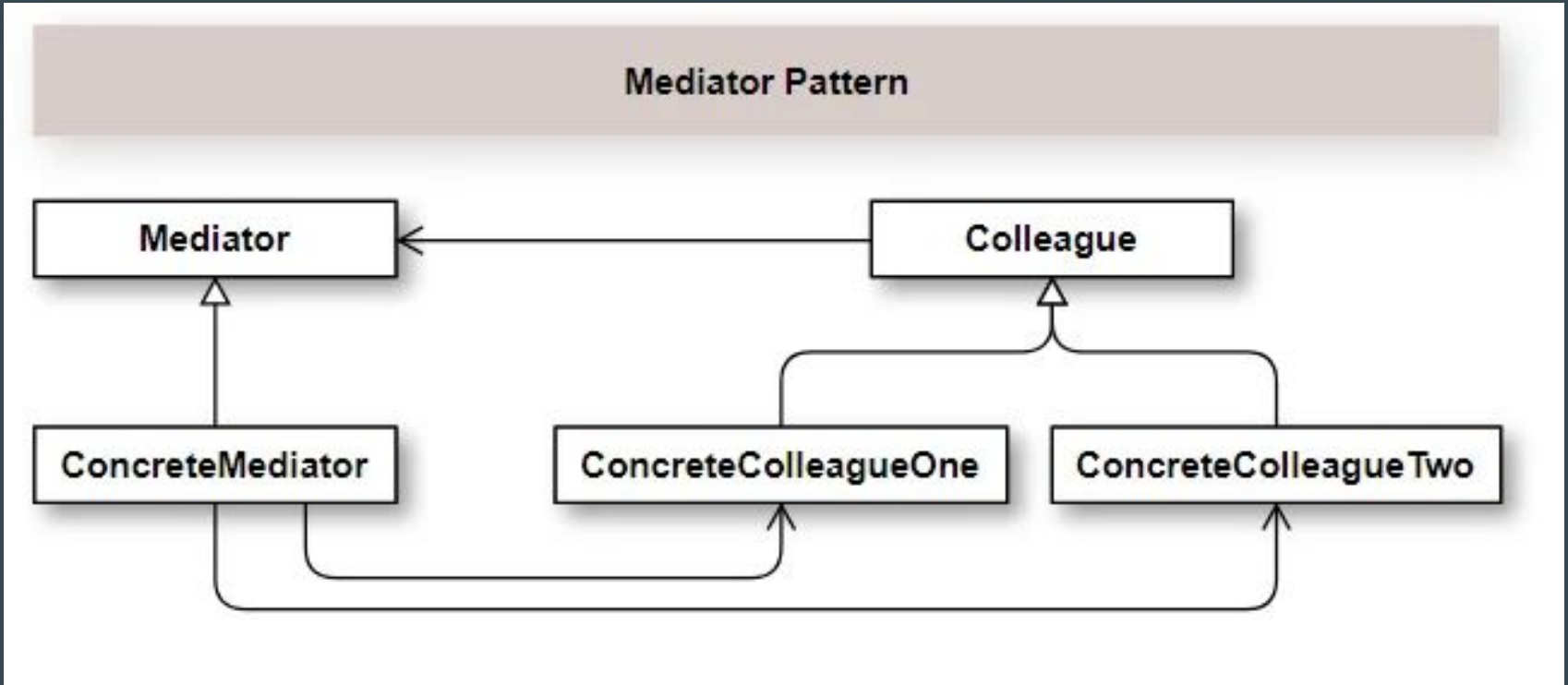
- **Exemplo de Contextualização:**
  - Em uma interface gráfica complexa, vários componentes de UI precisam se comunicar, resultando em um código difícil de manter e modificar.

Mediator - Solução proposta

- **Descrição:**
  - Introduz um objeto mediador que gerencia a comunicação entre os objetos.
  - Os objetos participantes se comunicam através do mediador, que encapsula a lógica de interação.

- **Diagrama de Classes:**
  - Mediator: Define uma interface para comunicação com os Colleagues.
  - ConcreteMediator: Implementa a interface Mediator e coordena a comunicação entre os Colleagues.
  - Colleague: Representa um objeto participante que se comunica através do Mediator.

Mediator - Diagrama de classes



Mediator - Quando Usar

Cenários Específicos:

- Comunicação complexa entre muitos objetos
- Sistemas GUI (Interface Gráfica do Usuário)

Condições Justificáveis:

- Necessidade de simplificação de comunicação
- Desejo de centralizar a lógica de interação

Mediator - Exemplo Prático

https://refactoring.guru/pt-br/design-patterns/mediator/java/example

Mediator - Vantagens e Desvantagens

Vantagens:

- Reduz o acoplamento entre colegas
- Facilita a manutenção e extensão

Desvantagens:

- Pode introduzir um ponto único de falha
- Mediador pode se tornar complexo

Conclusão

Sumário dos Pontos Principais:

- Observer: Notificação de múltiplos objetos
- Strategy: Intercâmbio de algoritmos
- Mediator: Centralização da comunicação

Reflexões Finais:

- Importância na simplificação e organização do código
- Facilitação da manutenção e evolução do sistema