Arquitetura de Software

Padrões de Projetos – Padrões Comportamentais

Nairon Neri Silva

Sumário

Padrões de Projeto Comportamentais

1. *Chain of Responsibility*
2. *Command*
3. *Interpreter*
4. *Iterator*
5. ***Mediator***
6. ***Memento***
7. *Observer*
8. *State*
9. *Strategy*
10. *Template Method*
11. *Visitor*

Mediator

Intenção

- Definir um objeto que encapsula a forma como um conjunto de objetos interage

- O *Mediator* promove o acoplamento fraco ao evitar que os objetos se refiram uns aos outros explicitamente e permite variar suas interações independentemente

Motivação

- O projeto orientado a objetos encoraja a distribuição de comportamento entre vários objetos
  - Pode resultar em muitas conexões entre objetos
  - Cada objeto tem o conhecimento muitos objetos

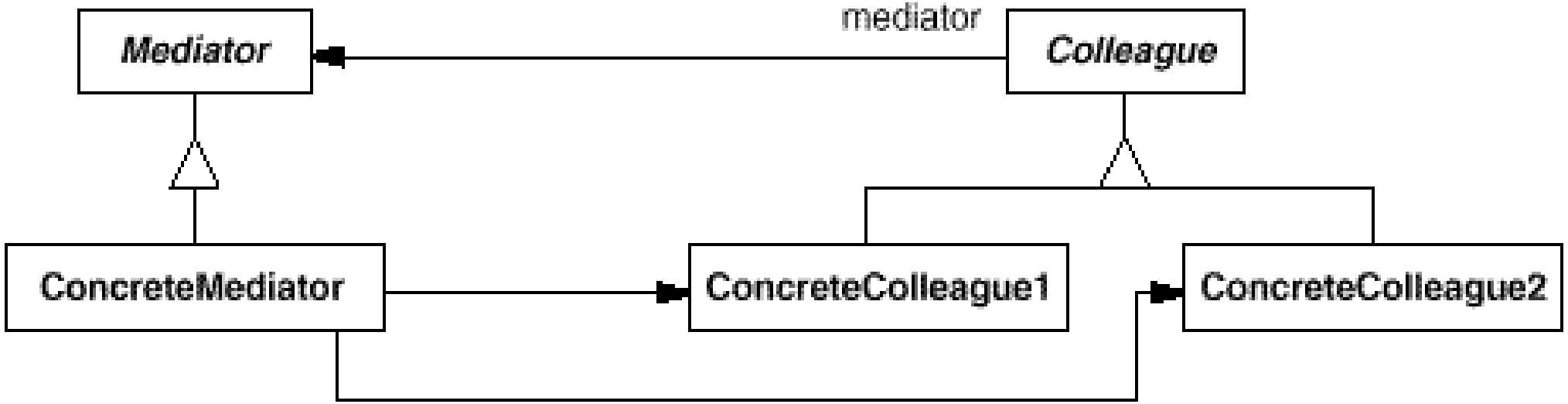
- Como exemplo, considere a implementação de caixas de diálogo em uma GUI
  - Uma caixa de diálogo usa uma janela pra apresentar uma coleção de componentes– botões, menus, entradas, etc.
  - Frequentemente existe dependência entre os componentes e a caixa de diálogo

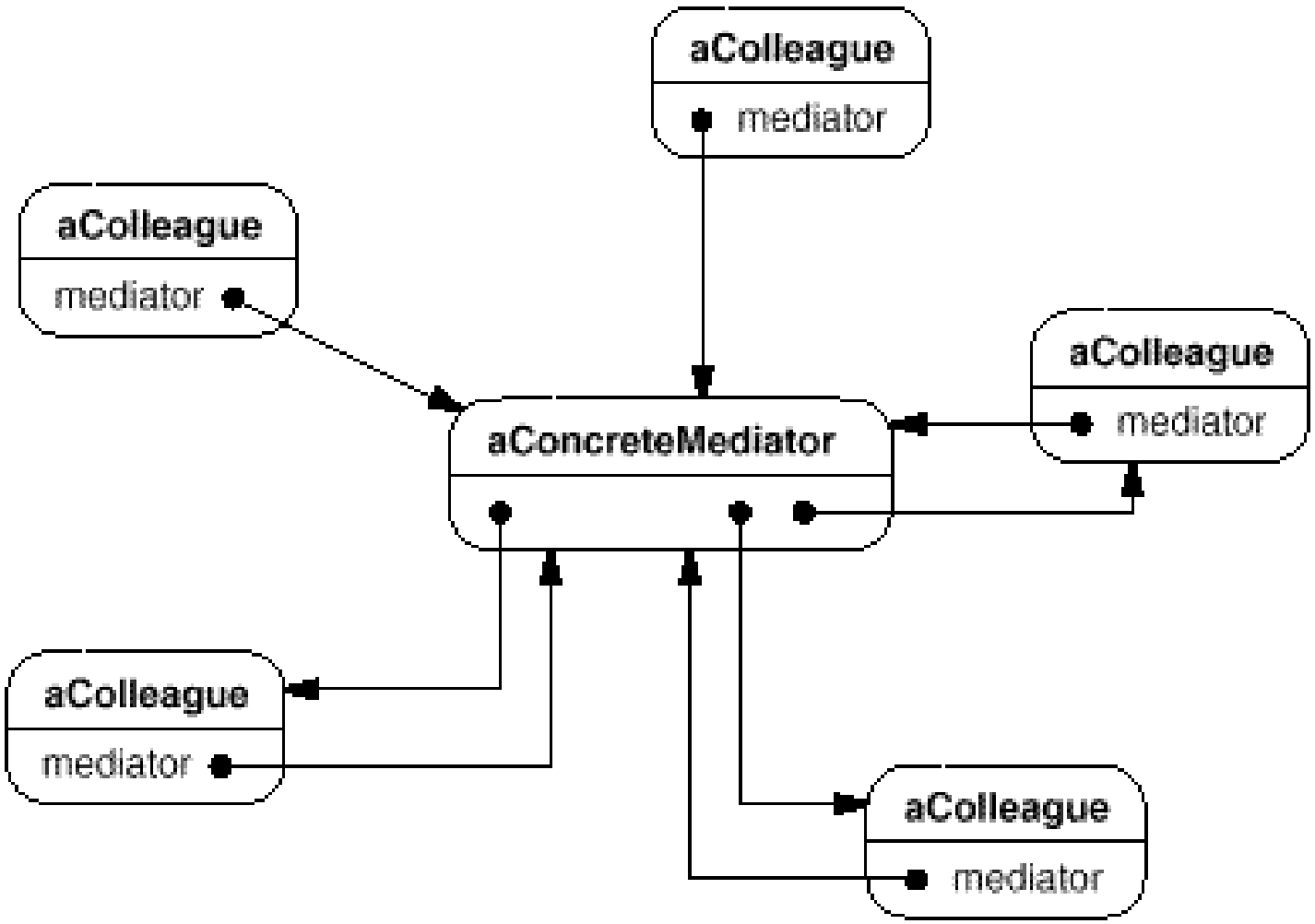
Motivação

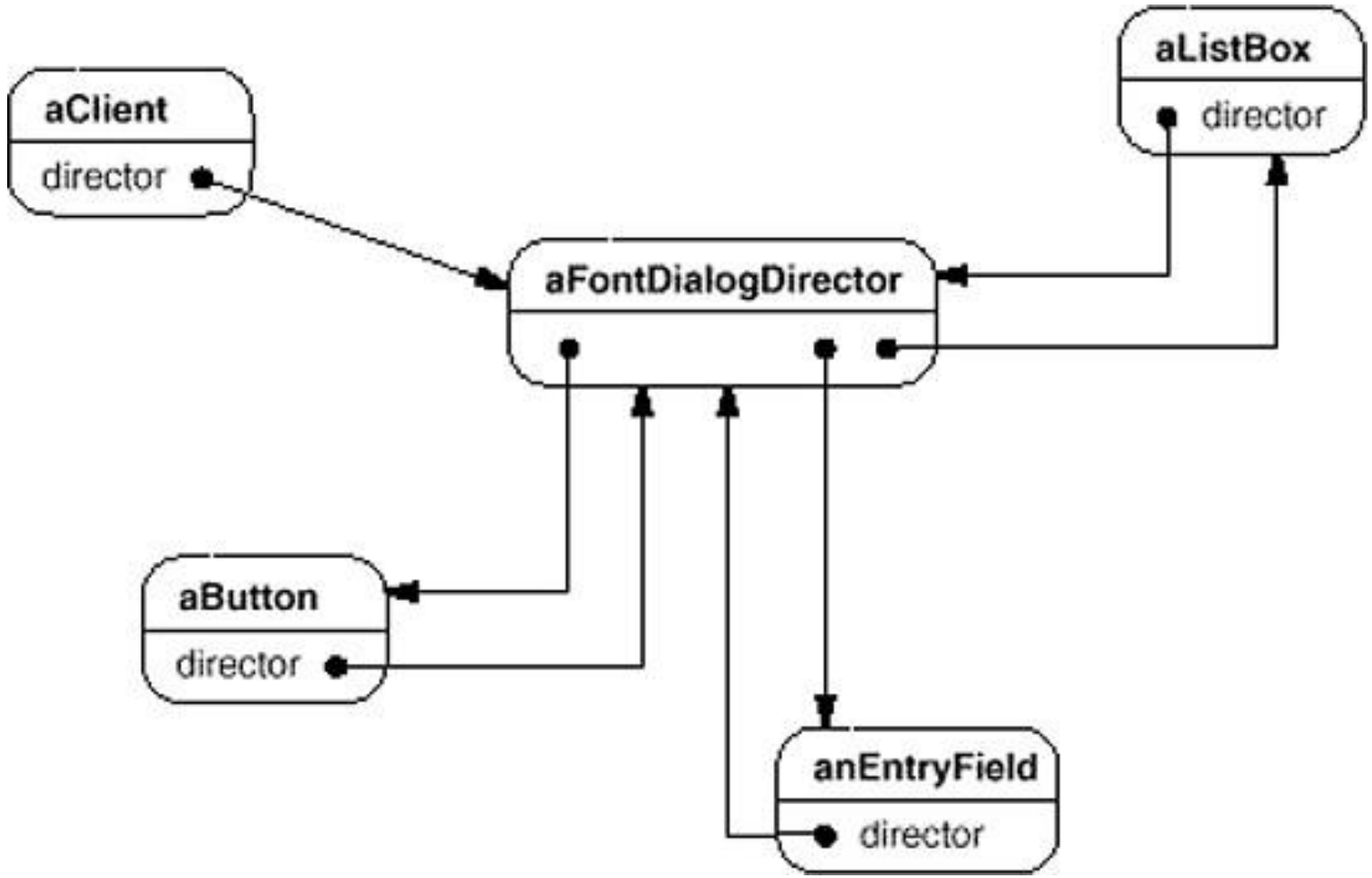
- Continuando...
  - Por exemplo, um botão pode se ficar desabilitado enquanto um campo de entrada permanecer vazio
  - Diferentes caixas de diálogo terão diferentes dependências entre componentes
  - Embora diálogos exibam os mesmos componentes, não podem reutilizar as classes – pois, devem ser customizadas para refletir as dependências específicas
  - Esse problema poderia ser evitado encapsulando o comportamento coletivo num objeto *Mediator*

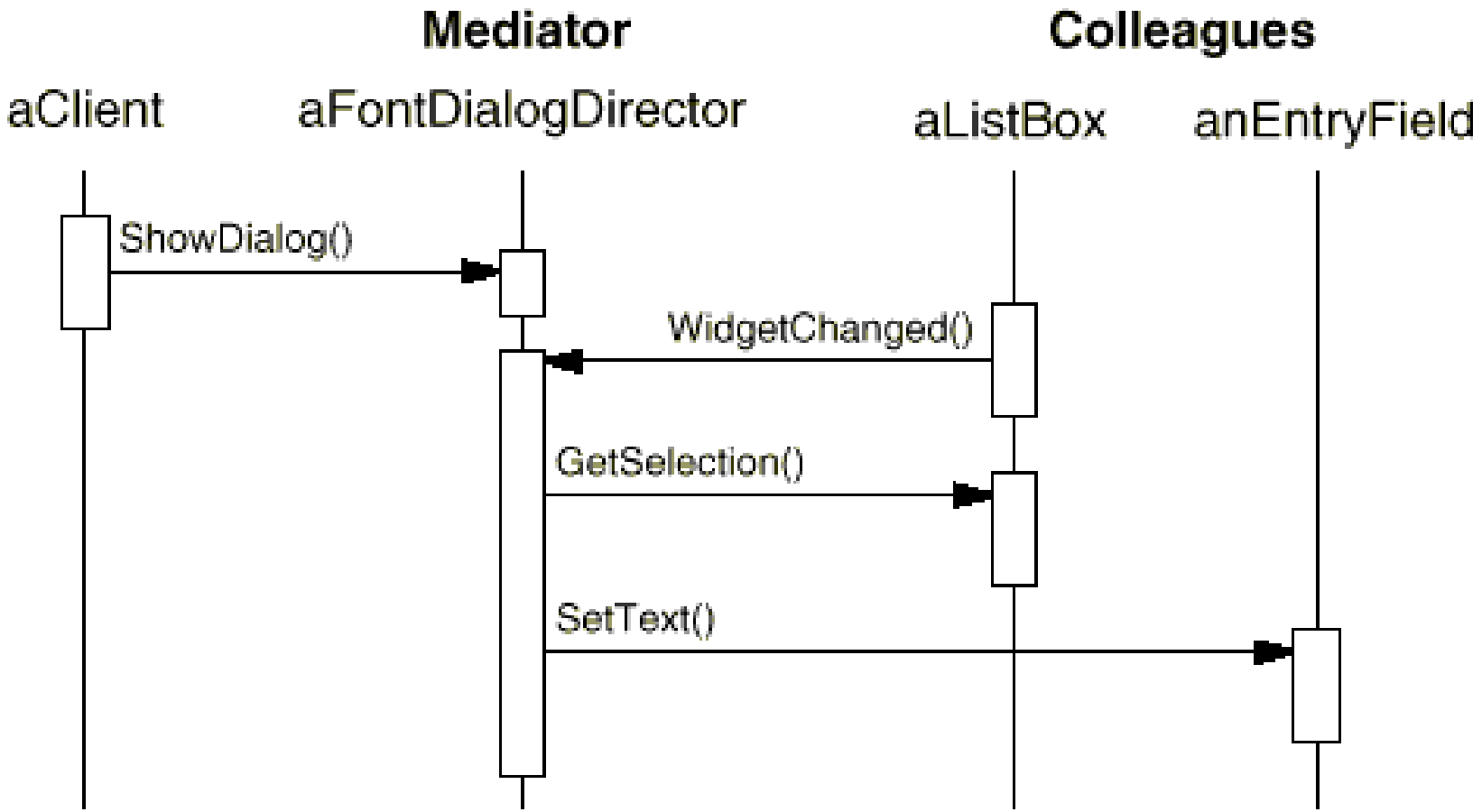
Aplicabilidade

- Utilize o padrão *Mediator* quando:
  1. um conjunto de objetos se comunica de maneiras bem-definidas, porém complexas –interdependência
  2. a reutilização de um objeto é difícil porque ele referencia e se comunica com muitos outros objetos
  3. um comportamento que está distribuído entre várias classes deveria ser customizável, ou adaptável, sem excessiva especialização em subclasses


Mediator
mediator
Colleague
ConcreteMediator
ConcreteColleague1
ConcreteColleague2


aColleague
mediator
aColleague
mediator
aConcreteMediator
aColleague
mediator
aColleague
mediator
aColleague
mediator


aClient
director
aListBox
director
aFontDialogDirector
aButton
director
anEntryField
director


Mediator
Colleagues
aClient
aFontDialogDirector
aListBox
anEntryField
ShowDialog()
WidgetChanged()
GetSelection()
SetText()

Exemplo/Participantes

1. ***Mediator (DialogDirector)***
   - Define a interface para comunicação com os *Colleagues*
2. ***ConcreteMediator (FontDialogDirector)***
   - Implementa o comportamento cooperativo – coordenação dos objetos *Colleagues*
3. ***Classes Colleague (ListBox, EntryField)***
   - Cada um conhece o seu objeto *Mediator*
   - Sempre que necessita se comunicar um *Colleague*, o faz através do *Mediator*

Consequências

1. Limita o uso de subclasses
2. Desacopla classes ”colegas”
3. Simplifica o protocolo dos objetos
4. Abstrai a maneira como os objetos cooperam
5. Centraliza o controle

Exemplo Prático

1. Imagine uma rede de telefonia celular... aparentemente você comunica diretamente com outra pessoa, mas o que acontece de fato é que você comunica com a operadora e essa repassa para o destinatário as informações.
2. Nesse caso a operadora é o Mediator, responsável por registrar (conhecer) todos os usuários da rede e intermediar a comunicação entre eles.
3. Vamos ver esse exemplo em Java

Exemplo baseado no projeto:
https://github.com/rohitksingh/Design_Patterns_aka_Hawai_Baatein/tree/master/src/MediatorDesignPattern

Saiba mais...

- https://www.youtube.com/watch?v=fb7NrdCo4Ko
- https://refactoring.guru/pt-br/design-patterns/mediator
- https://github.com/rohitksingh/Design_Patterns_aka_Hawai_Baatein/tree/master/src/MediatorDesignPattern

Acesse os endereços e veja mais detalhes sobre o padrão Mediator.

Memento

Também conhecido como ***Token***

Intenção

- Sem violar o encapsulamento, capturar e externalizar um estado interno de um objeto, de maneira que o objeto possa ser restaurado para este estado mais tarde

Motivação

- Algumas vezes é necessário registrar o estado de um objeto – implementação de *checkpoints*

- As informações de estado devem ser salvas em algum lugar, que modo que seja possível restaurar estados prévios de um objeto

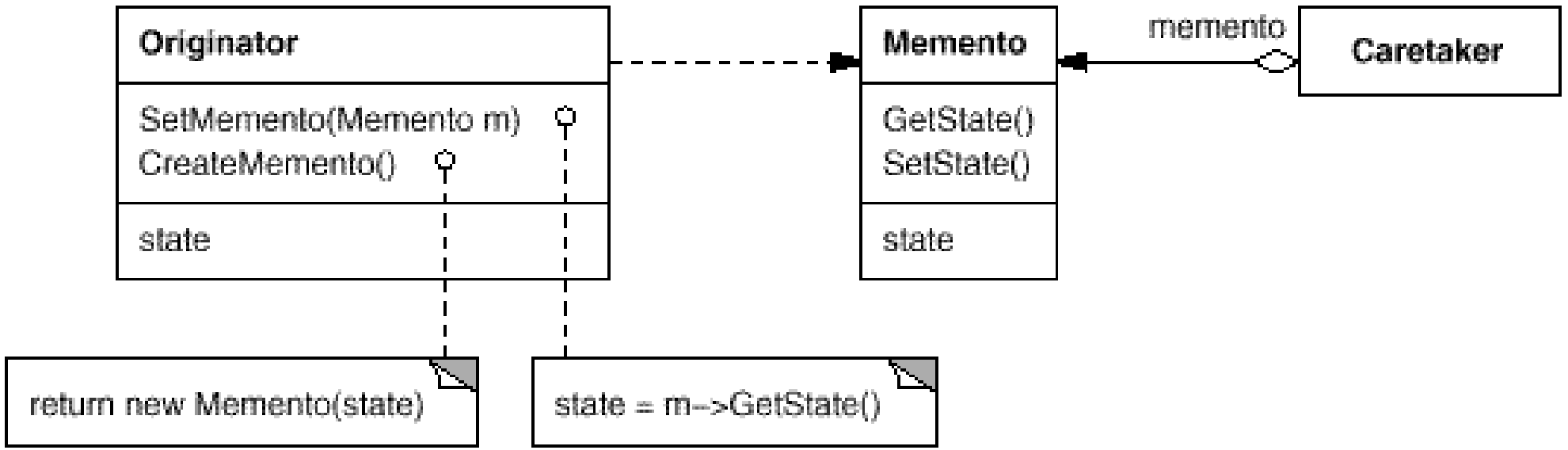
- Problema: possível violação do encapsulamento

Motivação

- Exemplo: considere um editor gráfico que suporta a conectividade entre objetos
  - um usuário pode conectar dois retângulos com uma linha, os quais permanecem conectados quando movidos
  - manter relações de conectividade entre objetos pode ser feita por meio de um sistema de solução de restrições
  - A interface desse *ContraintSolver* pode ser insuficiente para permitir a reversão precisa dos seus efeitos sobre outros objetos
  - Esse problema pode ser resolvido com o padrão *Memento*, o qual armazena um instantâneo do estado interno de outro objeto – **originador** do *Memento*

Aplicabilidade

- Utilize o padrão *Memento* quando:
  1. Um instantâneo (de alguma porção) do estado de um objeto deve ser salvo de maneira que possa ser restaurado para esse estado mais tarde
  2. Uma interface direta para obtenção do estado exporia detalhes de implementação e romperia o encapsulamento do objeto


Originator
SetMemento(Memento m)
CreateMemento()
state
Memento
GetState()
SetState()
state
memento
Caretaker
return new Memento(state)
state = m->GetState()

Exemplo/Participantes

1. ***Memento (SolverState)***
   - Armazena o estado interno do objeto *Originator*
   - Protege contra acesso por objetos que não o originador
   - O *Caretaker* vê uma interface restrita do *Memento*
2. ***Originator (ConstraintSolver)***
   - Cria um *Memento* contendo um instantâneo do seu estado interno corrente
   - Usa o *Memento* para restaurar o seu estado interno
3. ***Caretaker (undo mechanism)***
   - Responsável pela custódia do *Memento*
   - Nunca opera ou examina os conteúdos de um *Memento*

Consequências

1. Preservação das fronteiras de encapsulamento

2. Ele simplifica o originador

3. O uso de Mementos pode ser computacionalmente caro

4. Custos ocultos na custódia de Mementos. Quanto mais backups, maior será o consumo de memória

Exemplo Prático

1. Imagine uma aplicação que precisa armazenar os estados de alguns objetos e ao longo da execução vai salvando o último estado configurado.
2. Em um determinado momento é preciso identificar qual é o último estado salvo, pois essa informação é importante.
3. Vamos ver como ficará essa implementação em Java.

Exemplo baseado no site: https://sourcemaking.com/design_patterns/memento/java/1

Saiba mais...

- https://www.youtube.com/watch?v=crmLq8_FtLc
- https://refactoring.guru/pt-br/design-patterns/memento
- https://sourcemaking.com/design_patterns/memento/java/1

Acesse os endereços e veja mais detalhes sobre o padrão Memento.