# 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*

# *Observer*

Também conhecido como ***Publish-Subscribe***

# 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

# Motivação

- Um efeito colateral comum da modularização de um sistema é a necessidade de manter a consistência entre os objetos relacionados
  - Com baixo acoplamento e alta reusabilidade

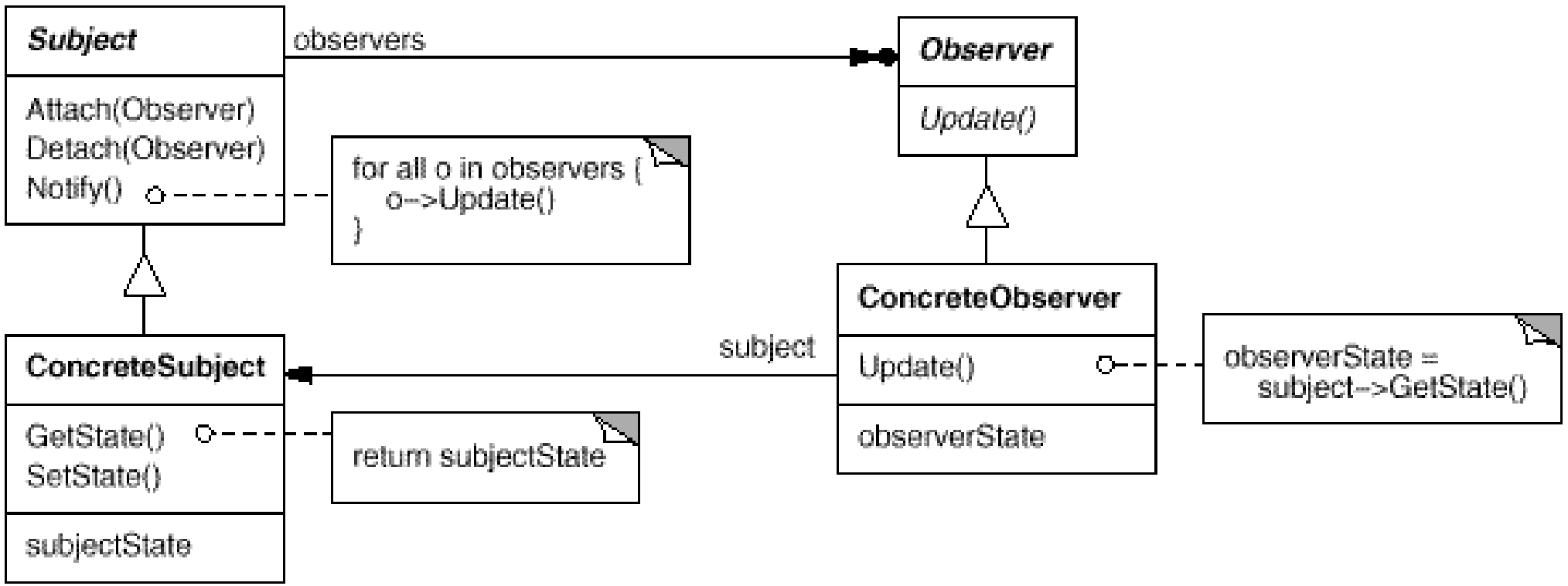
- Exemplo: dado um framework para construção de interfaces gráficas com o usuário
  - Separam os aspectos de apresentação da interface dos dados da aplicação
  - As classes que definem os dados da aplicação e apresentação podem ser reutilizadas independentemente

# Motivação

- Continuando...
  - Mas, uma ”planilha” pode trabalhar em conjunto com um ”gráfico de barras” – onde a mudança do primeiro pode ser refletida no segundo
    - Continuando independentes
- O padrão *Observer* descreve como estabelecer esses relacionamentos
  - Objetos ”observadores” se inscrevem para observar um dado objeto e são notificados quando mudanças de estado ocorrem

# Aplicabilidade

- Utilizaremos o padrão *Observer* quando:
  1. uma abstração tem dois aspectos, um dependente de outro, mas encapsulados em objetos separados (independentes)
  2. uma mudança em um objeto exige mudanças em outros, e você não sabe quantos objetos necessitam ser mudados
  3. um objeto deveria ser capaz de notificar outros objetos sem fazer hipóteses, ou usar informações, de que objetos são estes


Subject
observers
Observer
Update()
Attach(Observer)
Detach(Observer)
Notify()
for all o in observers {
o->Update()
}
ConcreteObserver
subject
ConcreteSubject
Update()
observerState =
subject->GetState()
GetState()
SetState()
return subjectState
observerState
subjectState

# Participantes

1. ***Subject***
   - Conhece os seus observadores (podem ser vários)
   - Fornece uma interface para acrescentar e remover objetos para associar e desassociar objetos observadores
2. ***Observer***
   - Define uma interface de atualização para objetos que deveriam ser notificados sobre mudanças em um dado objeto (observado)

# Participantes

3. ***ConcreteSubject***
   - Armazena estados de interesse para objetos *ConcreteObserver*
   - Envia uma notificação para os seus observadores quando seu estado muda
4. ***ConcreteObserver***
   - Mantém uma referência para um objeto *ConcreteSubject*
   - Armazena estados que deveriam permanecer consistentes com os do *Subject*
   - Implementa a interface de atualização de *Observer*, para manter o seu estado consistente com o do *Subject*

# Consequências

1. Permite a variação de observadores e observados de forma independente

2. Acoplamento abstrato entre *Subject* e *Observer*

3. Suporte para a comunicação em *broadcast*

4. Atualizações inesperadas ou desnecessárias

# Exemplo Prático

1. Precisamos de uma aplicação que faça a conversão de um número em base decima para várias outras bases (binária, octal, hexadecimal...).
2. Todas as conversões devem ser realizadas através de uma única chamada no programa principal.
3. Outras conversões podem ser adicionadas ou retiradas ao longo do tempo.

Exemplo baseado no site: https://www.tutorialspoint.com/design_pattern/observer_pattern.htm

# Saiba mais...

- https://www.youtube.com/watch?v=iMV1aHaijhQ
- https://refactoring.guru/pt-br/design-patterns/observer
- https://www.tutorialspoint.com/design_pattern/observer_pattern.htm

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

# *State*

***Objects for State***

# Intenção

- Permite a um objeto alterar o seu comportamento quando o seu estado interno muda

- O objeto parecerá ter mudado a sua classe

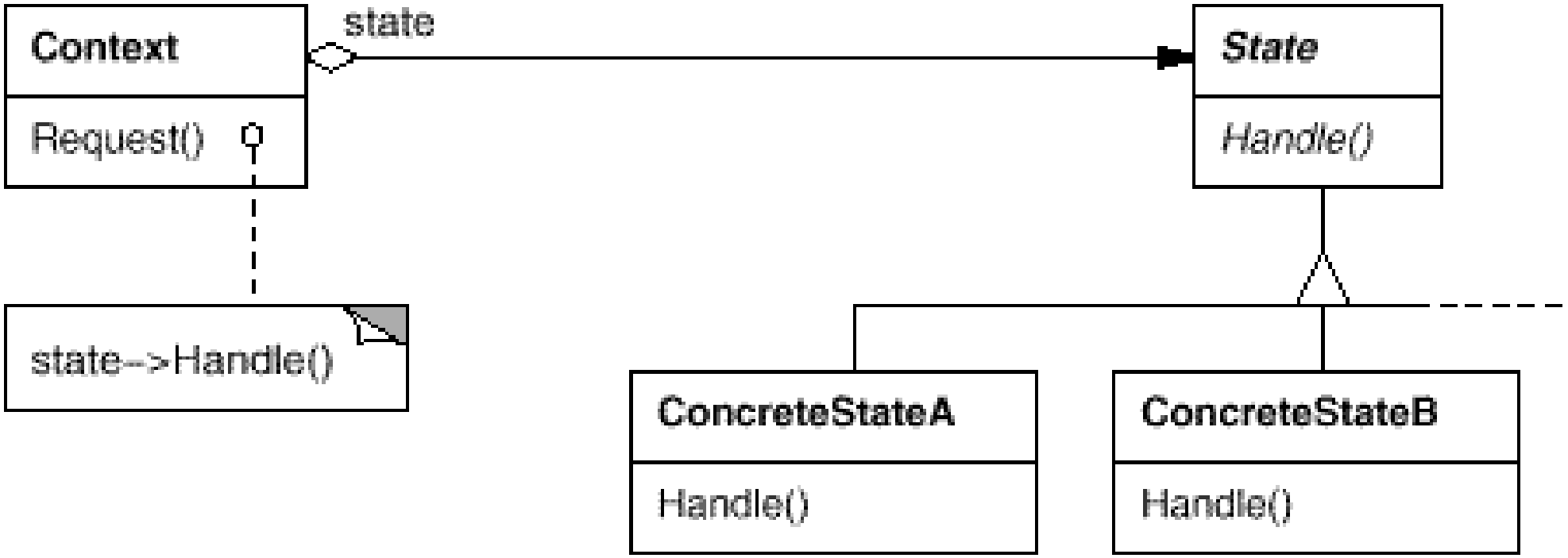
# Motivação

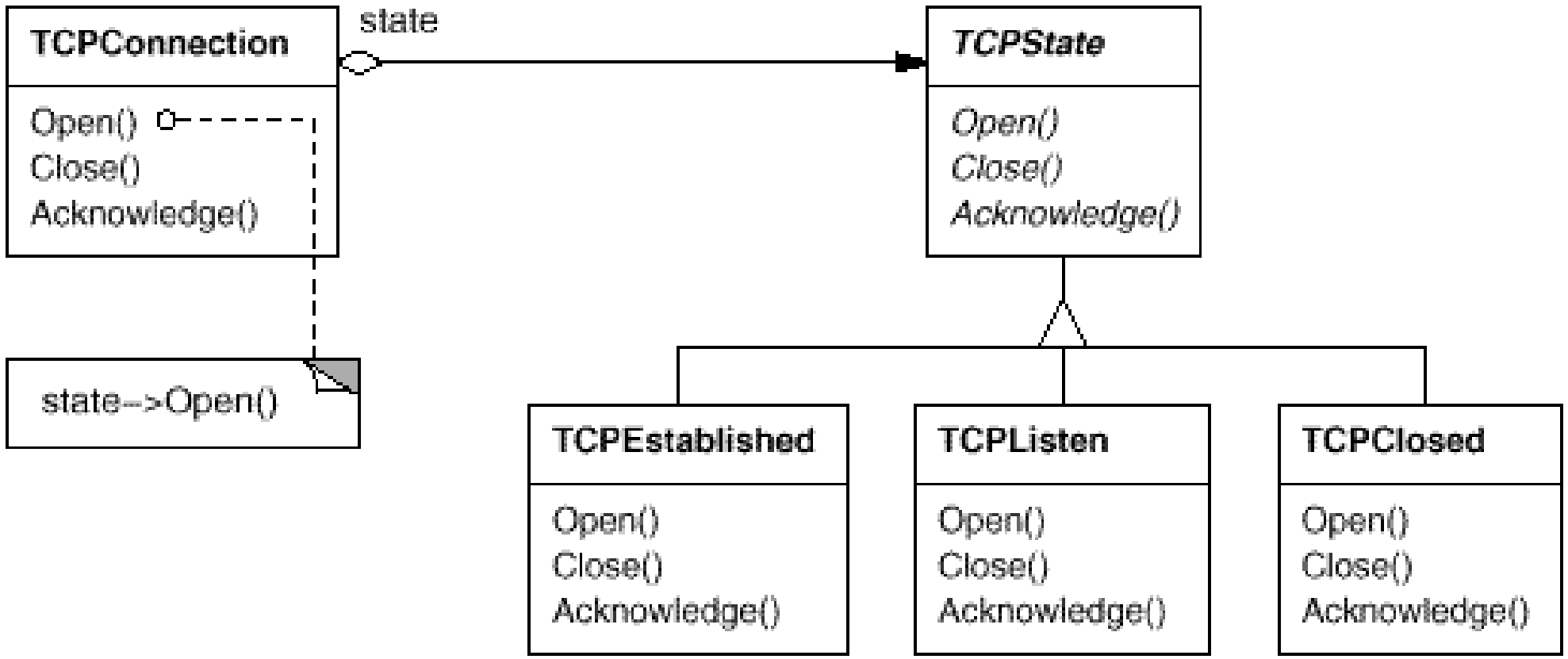
- Ex.: classe **TCPConnection –** pode estar em diferentes estados: estabelecida (*established*), escutando (*listening*), fechada (*closed*) e etc.
  - O efeito de uma solicitação depende do seu estado
  - Ex.: uma solicitação para abrir conexão (*open*)

- Ideia do padrão é introduzir uma classe abstrata (interface comum) para representar os estados

# Aplicabilidade

- Use nos seguintes casos:
  - O comportamento de um objeto depende do seu estado, o qual pode mudar em tempo de execução
  - Operações são construídas com muitos condicionais, que dependem do estado do objeto (*flags*)


Context
Request()
state
State
Handle()
state->Handle()
ConcreteStateA
Handle()
ConcreteStateB
Handle()


TCPConnection
Open()
Close()
Acknowledge()
state
state->Open()
TCPState
Open()
Close()
Acknowledge()
TCPEstablished
Open()
Close()
Acknowledge()
TCPListen
Open()
Close()
Acknowledge()
TCPClosed
Open()
Close()
Acknowledge()

# Exemplo/Participantes

1. **Context (TCPConnection)**
   - Define a interface de interesse para os clientes
   - Mantém referência a uma subclasse de **ConcreteState**
2. **State (TCPState)**
   - Define a interface relativa a um estado específico
3. Subclasses **ConcreteState (TCPEstablished, TCPListen, TCPClosed)**
   - Cada subclasse implementa o comportamento associado ao estado correspondente

# Consequências

1. Agrupa comportamentos específicos de estados e separa comportamentos para estados diferentes

2. Torna explícitas as transições de estado

3. Objetos **State** podem ser compartilhados

# Exemplo Prático

1. Neste exemplo, o padrão State permite que os mesmos controles do tocador de mídia se comportem de maneira diferente, dependendo do estado atual da reprodução.
2. A classe principal do tocador contém uma referência a um objeto de estado, que executa a maior parte do trabalho para o tocador.
3. Algumas ações podem acabar substituindo o objeto de estado por outro, o que altera a maneira como o tocador reage às interações do usuário.

Exemplo baseado no site: https://refactoring.guru/pt-br/design-patterns/state/java/example

# Saiba mais...

- https://www.youtube.com/watch?v=tSTPS2oHDmw
- https://refactoring.guru/pt-br/design-patterns/state

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