AULA 14 – PADRÃO OBSERVADOR

GSI020 – Programação Orientada a Objetos II

Prof. Dr. Murillo G. Carneiro
mgcarneiro@ufu.br

UFU

Material baseado nos slides cedidos pelo Prof. Rafael D. Araújo (FACOM/UFU)

Objetivo da aula

- Entender o funcionamento do padrão de projeto *Observer (Observador)*.

Motivação

- Imagine um sistema com inúmeras classes que cooperam entre si. As alterações dos objetos de tais classes devem ser consistentes em todas elas.
- Como resolver? Deixá-las dependentes umas das outras?
  - *Não! Isso aumenta o acoplamento e diminui a reusabilidade!*
- Imagine o cenário de uma assinatura de *feeds* de notícias. Uma pessoa pode assinar inúmeros feeds. Ao incluir uma nova notícia, como notificar cada pessoa?

Motivação

- Imagine o cenário de uma assinatura de *feeds* de notícias. Uma pessoa pode assinar inúmeros feeds. Ao incluir uma nova notícia, como notificar cada pessoa?

Observador

- É um padrão de projeto de propósito **comportamental** com escopo de **objetos**.
- Propõe uma solução para que as alterações ocorridas em um objeto possam notificar outros objetos, desencadeando ações necessárias.
- Também chamado de *publish-subscribe*.

Intenção

- Definir uma dependência *um-para-muitos* entre objetos, de maneira que quando um objeto em particular (*observado*) muda de estado, todos os seus dependentes (*observadores*) são notificados e atualizados automaticamente.

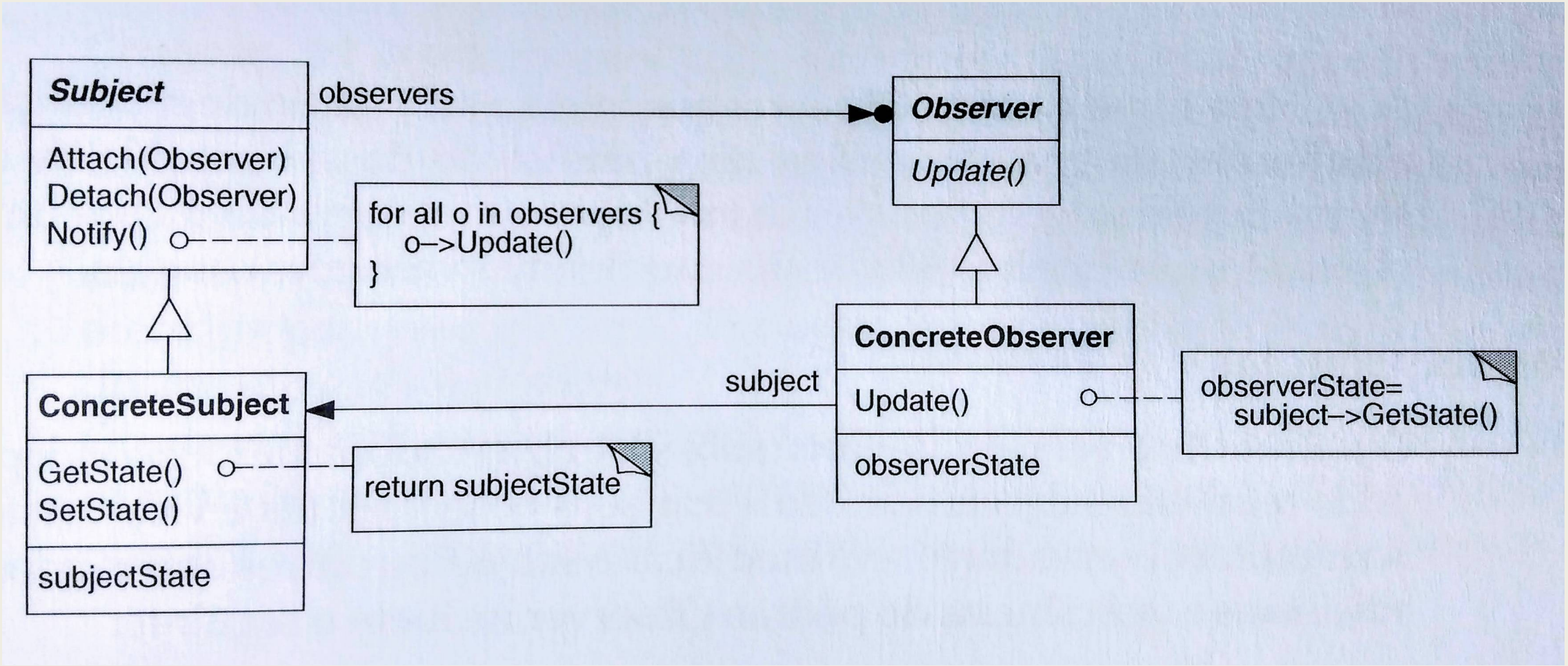
Quando usar

- Imagine que uma determinada abstração possui duas características dependentes uma da outra. Encapsulando essas características em objetos separados, é possível variá-los e reutilizá-los independentemente.
- Quando uma mudança em um objeto exige mudanças em outros e o objeto que sofreu alteração não conhece os outros objetos que devem ser alterados.

Elementos participantes

- **Subject**: representa uma abstração do sujeito a ser observado e fornece uma interface para adicionar e remover objetos *Observer*.
- **Observer**: define uma interface de atualização para objetos que devem ser notificados sobre mudanças em um *Subject*.
- **ConcreteSubject**: implementa uma classe de sujeitos concretos e armazena os estados que interessam ao *ConcreteObserver*. É responsável por notificar as alterações aos *Observers*.
- **ConcreteObserver**: mantém uma referência para um objeto ConcreteSubject e implementa a interface de atualização definida em *Observer*, de forma a executar ações necessárias quando notificado.

Estrutura


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

Benefícios

- Evita que um objeto precise monitorar constantemente o estado de outro(s) objeto(s).
- Permite adicionar/remover objetos observados e observadores de forma independente.
- Diminui o acoplamento.
  - *A única coisa que o sujeito sabe sobre um observador é que ele implementa uma certa interface (observer).*

Desvantagens

- Atualizações inesperadas
  - *Um observador não tem conhecimento da presença dos outros, o que pode levar a uma cascata de atualizações que podem causar alterações indevidas*

Problema prático

- Imagine o cenário de uma estação meteorológica que possui dois sensores: de temperatura e de umidade.
- Dependendo dos valores da temperatura e da umidade, um cálculo de previsão de tempo é realizado.
  - *mínima = temperatura atual – 10% da umidade*
  - *máxima = temperatura atual + 10% da umidade*
- Considere duas telas: (1) mostra a temperatura e a umidade atual e (2) mostra a previsão do tempo (baseado no cálculo realizado).

Problema prático

- Identifique o que seria os elementos do padrão observador
  - *Subject*
  - *ConcreteSubject*
  - *Observer*
  - *ConcreteObserver*

Problema prático

- Identifique o que seria os elementos do padrão observador
  - *Subject => EstacaoSubject*
  - *ConcreteSubject => DadosTempo*
  - *Observer => TelaObserver*
  - *ConcreteObserver => TelaTempoAgora e TelaPrevisao*

Referências

- GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1995. Capítulo 5.