Observer 💻

Análise e Desenvolvimento de Sistemas - 6º Período

Observer

- Nos dias atuais é bastante comum o conceito de **observabilidade (não relacionado ao padrão)**, isto é, monitoramento
- Aplicamos monitoramento em diferentes aspectos das nossas aplicações e infraestrutura
- Um exemplo é o **Prometheus**. Solução para monitoramento de aplicações e infra que oferece uma plataforma para salvar os dados e geração de alertas

Observer

- Considere que você possua diferentes microserviços sendo hospedados em um cluster
- Monitorar a saúde desse cluster é importante para manter uma alta disponibilidade
- MTTR (Tempo médio de recuperação); MTBF (Tempo médio entre falhas)
- Queremos que essas médias sejam sempre pequenas

Observer

- Para isso, precisamos garantir a geração de alertas, ou notificações
- Uma notificação nada mais é que um alerta gerado quando **um evento de interesse acontece**
- Esse cenário pode ser implementado dentro do nosso código orientado a objetos
- O padrão que permite isso é o **observer**, também conhecido como **listener**

Observer

- Implementa um mecanismo de **assinatura** onde podemos notificar objetos sobre eventos que aconteçam com o objeto observado
- Nosso objeto observado é "verificado", ou "escutado", a todo momento pelos outros objetos, até que um evento de interesse aconteça

Observer

- O objeto que é observado, também chamado de **Sujeito**, é quem possui o estado de interesse
- Aqui, vamos delegar a **responsabilidade de notificar as suas mudanças de estado para ele mesmo**
- Portanto, vamos chamá-lo de **publicador**. Os outros objetos que possuem interesse nele serão chamados de **assinantes**

Observer

- O publicador irá implementar um **mecanismo de assinatura**, onde os objetos interessados poderão assinar ou desassinar uma corrente de eventos do objeto publicador
- Para criar esse mecanismo, o publicador irá implementar uma lista para armazenar uma referência de seus objetos assinantes e outros métodos para adicionar e remover esses assinantes

Observer

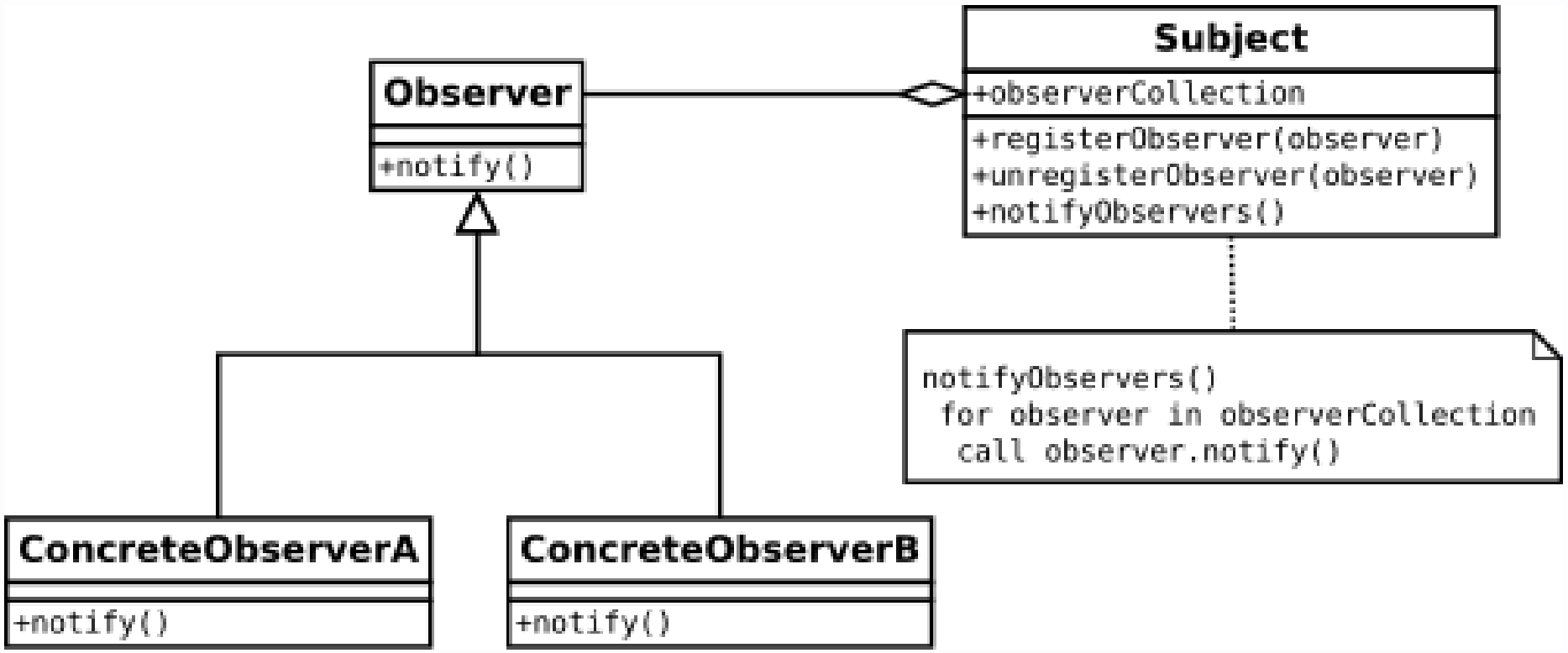
```
class Subject:
    def __init__(self):
        self.observers = []

    def add_observer(self, observer):
        self.observers.append(observer)

    def remove_observer(self, observer):
        self.observers.remove(observer)
```

Observer

- Para garantir que o objeto publicador consiga interagir com todos os assinante, criamos uma interface ou classe pai onde os assinantes irão implementar ou herdar. Dessa maneira podemos definir todos os métodos necessários para o objeto publicador. Essa interface irá declarar o método de notificação.

Observer

- A aplicação também pode possuir mais de um publicador. Para isso criamos uma interface para todas as publicadoras implementarem


Observer
+notify()
Subject
+observerCollection
+registerObserver(observer)
+unregisterObserver(observer)
+notifyObservers()
notifyObservers()
for observer in observerCollection
call observer.notify()
ConcreteObserverA
+notify()
ConcreteObserverB
+notify()