UNIVERSIDADE FEDERAL DE GOIÁS
CAMPUS SAMAMBAIA
INSTITUTO DE INFORMÁTICA
PROGRAMAÇÃO ORIENTADA A OBJETOS
PROF. DIRSON SANTOS DE CAMPOS
PROFA. RENATA DUTRA BRAGA

PEDRO CÉZAR SILVA FERREIRA

ESTUDO DE CASO: NOÇÕES DE DESIGN PATTERNS

GOIÂNIA
2023

Observer Pattern

- **Problema**:

Quando há uma necessidade de notificar vários objetos sobre as mudanças em outro objeto sem acoplar fortemente o observador ao objeto observado.

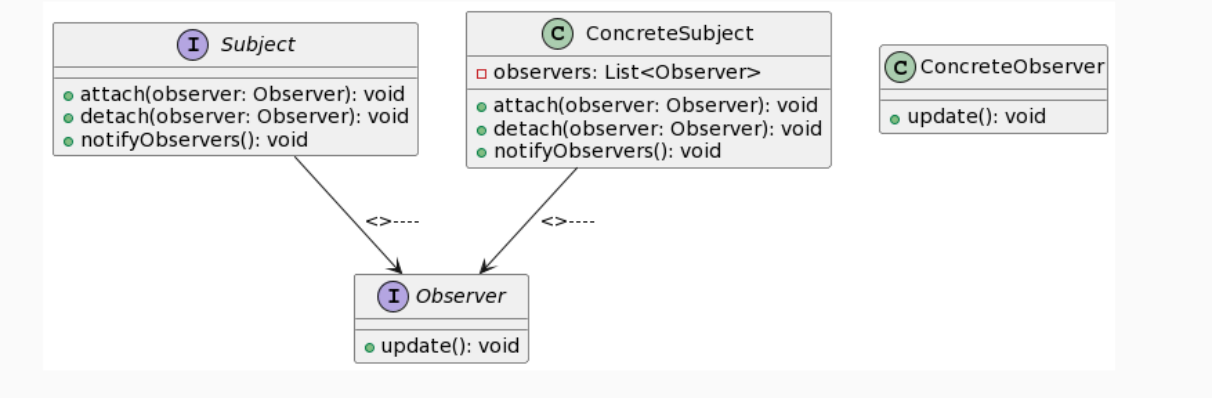
- **Contexto**:

É útil quando um objeto (sujeito) precisa informar outros objetos (observadores) sobre mudanças em seu estado, sem que os observadores precisem estar cientes da existência uns dos outros.

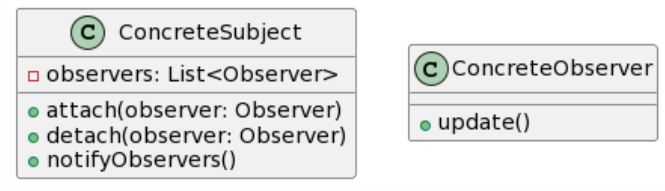
- **Solução**:

O padrão Observer define uma relação um-para-muitos entre objetos, onde, quando o estado de um objeto (sujeito) muda, todos os seus observadores são notificados automaticamente.

- **Diagrama de Classes**:



- **Classes e Métodos em Java**:




```java
// Subject (Sujeito)
public interface Subject {
    void attach(Observer observer);
```

```
    void detach(Observer observer);
    void notifyObservers();
}

// Observer (Observador)
public interface Observer {
    void update();
}

// ConcreteSubject (Sujeito Concreto)
public class ConcreteSubject implements Subject {
    private List<Observer> observers = new ArrayList<>();

    @Override
    public void attach(Observer observer) {
        observers.add(observer);
    }

    @Override
    public void detach(Observer observer) {
        observers.remove(observer);
    }

    @Override
    public void notifyObservers() {
        for (Observer observer : observers) {
            observer.update();
        }
    }
}

// ConcreteObserver (Observador Concreto)
public class ConcreteObserver implements Observer {
    @Override
    public void update() {
        // Lógica para lidar com a atualização do estado
    }
}
```

- **Referências**:

Design Patterns: Elements of Reusable Object-Oriented Software (E. Gamma, R. Helm, R. Johnson, J. Vlissides)

- **Exemplo de Código em Java**:

```java
public class ObserverExample {
    public static void main(String[] args) {
        ConcreteSubject subject = new ConcreteSubject();
        ConcreteObserver observer1 = new ConcreteObserver();
        ConcreteObserver observer2 = new ConcreteObserver();

        subject.attach(observer1);
        subject.attach(observer2);

        // Alguma mudança no estado do sujeito
        subject.notifyObservers();
    }
}
```

- **Explicação do Código Apresentado**:

Neste exemplo, ConcreteSubject é o sujeito que mantém uma lista de observadores. ConcreteObserver é o observador que implementa a lógica específica no método update(), que é chamado quando o sujeito notifica os observadores sobre uma mudança de estado. O exemplo no final mostra como criar um sujeito, adicionar observadores a ele e notificar os observadores quando há uma mudança no estado.