Observerpattern

Einsatzzweck

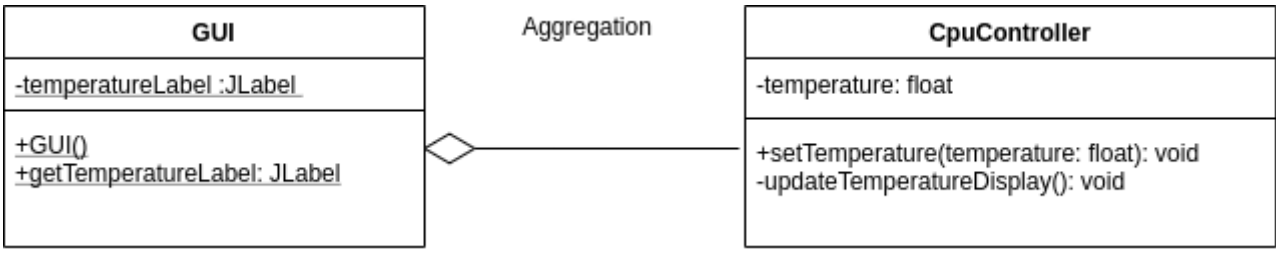
Es gibt unterschiedliche Wege wie Objekte die Änderung eines Ihrer Zustände (Attributwerte) kommunizieren können. Eines davon ist das ObserverPattern.

Anti-Pattern (Push-Methode)

in dem geänderten Hauptobjekt (CPU) wird das davon abhängige Objekt (Label) durch einen Methodenaufruf über die Änderung eines Zustandes (Temperatur) informiert.

- Das zu informierende Objekt (Label) muss bekannt sein und darf sich nicht mehr ändern.

```
class CpuController {
      private void updateTemperatureDisplay() {
     // Direkte Referenz auf eine GUI-Komponente
      JLabel temperatureLabel = MyGUIApp.getTemperatureLabel();
      temperatureLabel.setText("CPU Temperature: " + temperature +
            "°C");}….
```



In diesem Beispiel führt der `CpuController` direkte Aktualisierungen an der `JLabel`-Komponente in der GUI durch, indem er direkt auf die GUI-Komponente zugreift. Dies führt zu einer **starken Kopplung** zwischen der Logik (CpuController) und der Benutzeroberfläche (GUI).

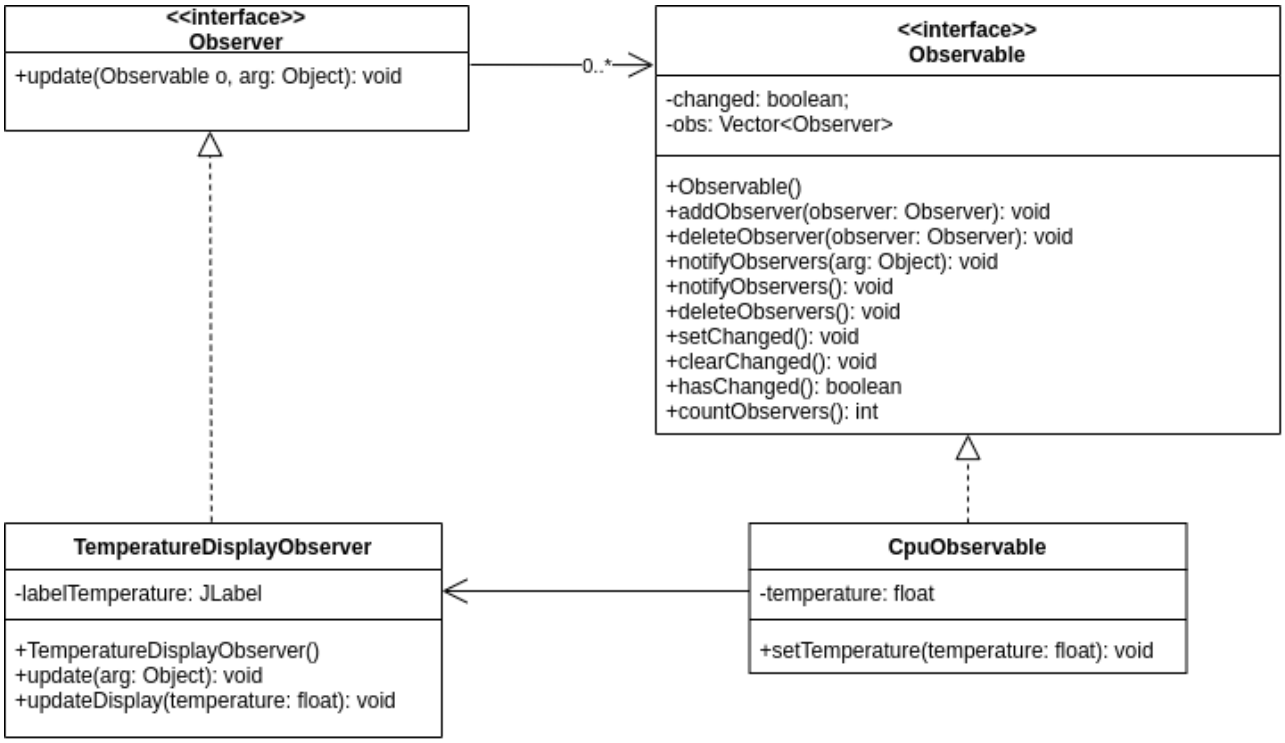
- Direkte Kopplung zwischen Subjekt und Objekt-Klassen.
- Schwierige Wartung und Erweiterung des Codes.

Besser: Observer-Pattern für eine lose Kopplung zwischen Subjekt und Observer, bzw. es dreht diese um

Observerpattern (publish/subscribe-Methode)

Die Abhängigkeiten sind umgekehrt.
Das zu informierende Objekt (Label) registriert seinen Wunsch beim Hauptobjekt (CPU-Temperatur) bei Änderungen benachrichtigt zu werden.
Ändert sich der Zustand bei dem Hauptobjekt (CPU-Temperatur) werden sie informiert.

```
Class TemperatureDisplay implements Observer {
    @Override
    public void update(Observable o, Object arg) {
      if (o instanceof CpuManager) {
        float newTemperature = (Float) arg;
        // Spezifische Logik für die Aktualisierung der Temperatur
           anzeige }}
}
```



INFO

- Mangel an Generizität und Typsicherheit. Prüfung mit instance of und cast
- Eingeschränkte Flexibilität und Erweiterbarkeit.

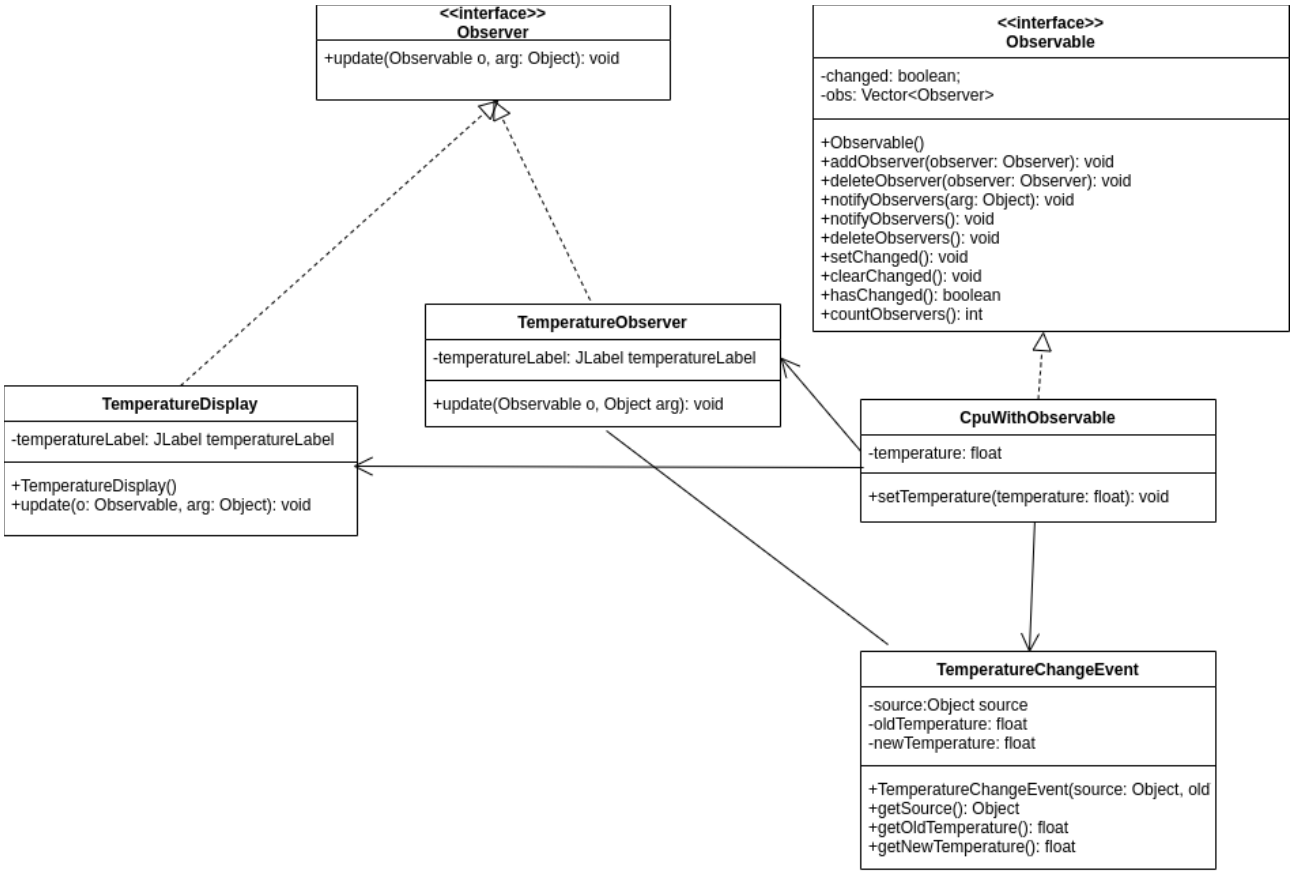
MERKE

- **Besser: Listener verwenden und im Eventhandler nicht prüfen, ob es “mein Event” ist**

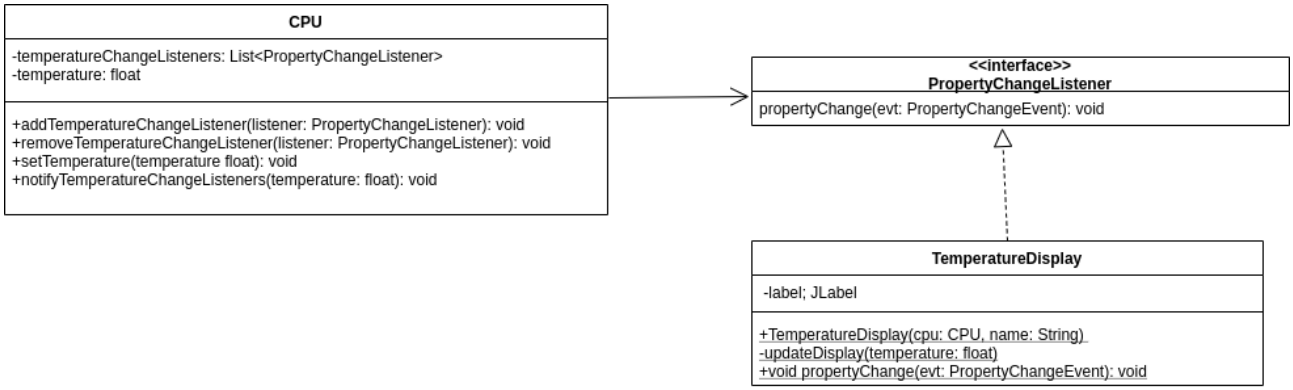
ObserverPattern mit Changemanager

Der Änderungsmanager ermöglicht es, Änderungen zu verfolgen und nach Bedarf weitere Reaktionen darauf zu initiieren.

In diesem Beispiel wird `CpuWithObservable` von `Observable` abgeleitet und benachrichtigt Observer-Objekte über Änderungen mit der Methode `notifyObservers`. `TemperatureChangeEvent` wird als Container für Informationen über Temperaturänderungen verwendet. Der `TemperatureObserver` implementiert die `Observer`-Schnittstelle und reagiert auf Änderungen, indem er die `update`-Methode implementiert.



PropertychangeListener




```
propertyChange(new PropertyChangeEvent(this, "temperature", oldTem
      perature, newTemperature));
```

INFO

- Typsicherheit durch Generizität.
- Flexibilität, ermöglicht das Abhören von Änderungen an spezifischen Eigenschaften.
- Bessere Wartbarkeit und Erweiterbarkeit des Codes.

MERKE

- **Verwenden Sie `PropertyChangeListener` für eine moderne und flexible Implementierung des Observer-Patterns.**

Quellen

Kecher, C. et al: „UML 2.5 – Das umfassende Handbuch“, 6., aktualisierte Auflage 2018, 1., korrigierter Nachdruck, Rheinwerk Computing, Bonn 2020
Geirhos, M.: „Entwurfsmuster – Das umfassende Handbuch“, 1. Auflage, Rheinwerk Verlag, Bonn 2015