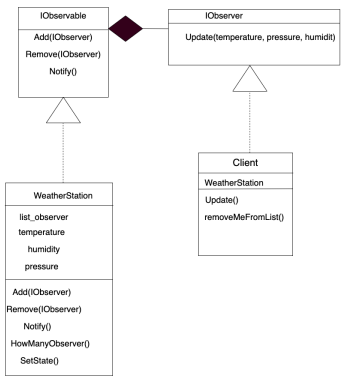
Observer design pattern

Karol Janik, Akadiusz Jędruch, Michał Kazanecki,
Krzysztof Kubień, Mikołaj Wielebnowski

21 czerwca 2021

1 Opis wzorca

Obserwator jest behawioralnym wzorcem projektowym, którego sktruktura opiera się na na obserwatorach (observers) i obiekcie obserwowanym (subject), gdzie obserwatorzy zostają poinformowani automatycznie o zmianie w obiekcie. Jego zastosowanie pozwala na dostęp do wszystkich zmian, bez ciągłego nasłuchiwania. Rozwiązuje on problem, gdzie ktoś zainteresowany nowymi informacjami w danym obiekcie mógłby ciągle sprawdzać wszystkie informacje jaki obiekt posiada, co prowadzi do sytuacji, gdzie zarówno obiekt jak i obserwator są bombardowane zapytaniami. Najprostszym rozwiązaniem jest rozesłanie przez obiekt informacji w momencie, gdy stanie się jakieś zdarzenie, jednak wymagana tutaj jest lista obserwatorów. Jego zastosowanie gwarantuje otrzymanie w jak najszybszy sposób informacji odnośnie zdarzenia do wszystkich zainteresowanych. Wzorzec wykorzysuje mechanizm subskrybcji w klasie obiektu publikującego, która pozwala ową subskrybcje rozpocząć lub przerwać. W momencie zdarzenia obiekt publikujący korzysta z dynamicznej listy subskrybentów i rozsyła im dane wybraną metodą.
Implementacja wzorca w naszym projekcie przedstawiona została na diagramie UML poniżej:



Rysunek 1: Struktura projektu.

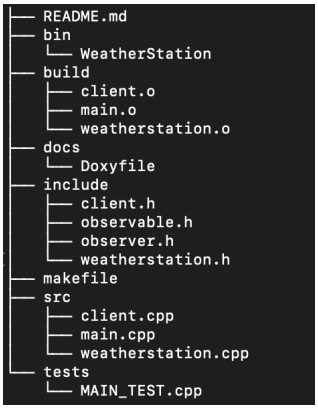
2 Opis projektu

Projekt WeatherStation wykorzystuje zadany wzorzec projektowy tworząc publikującego weatherstation, który posiada dynamiczną listę subskrybentów oraz funkcję publikującą oraz obserwatora w postaci client'a, który jest subskrybentem i zostaje poinformowany o zmianie.
Do pracy zespołowej wykorzystano plaformę github. Po utworzeniu programów utworzono plik makefile rozszerzany o kolejne funkcje. Kod został odpowiednio zakomentowany umożliwiając wygenerowanie automatycznej dokumentacji, co również wymagało utworzenie i dostosowanie pliku Doxyfile oraz funkcję testu korzystając z Google Test. Ostateczne opcje to:

1. *make*, by skompilować program.
2. *make run*, by uruchowmić i ewentualnie skompilować program.
3. *make docs*, by wygenerować i otrzworzyć dokumentację.
4. *make test*, by skompilować i wykonać test.
5. *make clean*, by usunąc pliki wykonywalne, folder build oraz pliki dokumentacji

Repozytorium projektu znajduje się stronie: github.com

2.1 Stuktura plików



Rysunek 2: Drzewko plików

2.2 Lista funkcji wraz z klasami

- Add() : IObservable , WeatherStation
- Client() : Client
- HowManyObserver() : WeatherStation
- Notify() : IObservable , WeatherStation
- Remove() : IObservable , WeatherStation
- removeMeFromList() : Client
- SetState() : WeatherStation
- setUp() : WeatherStationTest
- TearDown() : WeatherStationTest
- Update() : Client , IObserver
- ~IObserver() : IObserver

Rysunek 3: Drzewko plików

Literatura

[1] https://github.com/karol-janik/observer-pattern

[2] https://refactoring.guru/pl/design-patterns/observer

[3] https://en.wikipedia.org/wiki/Observer_pattern