Serie 3

Content:

- Design Pattern: Observer

(14) Design Pattern: Observer

Given a simple application implementing and "analog" clock, made of the following classes/interface in the `clock` package:

- The main class `ClockApp.java`.
- The utility class (Singleton) `clock.util.PositionManager.java` generating the window positions on the screen.
- The `clock.timer.ClockTimer.java` class updating the time.
- The `clock.analog.AnalogClock.java` frame containing a panel with the analog clock.
- The `clock.analog.AnalogClockPanel.java` class drawing the hands and numbers of the analog clock.

The class diagram of Figure 1 show the structure and the relations of these classes.

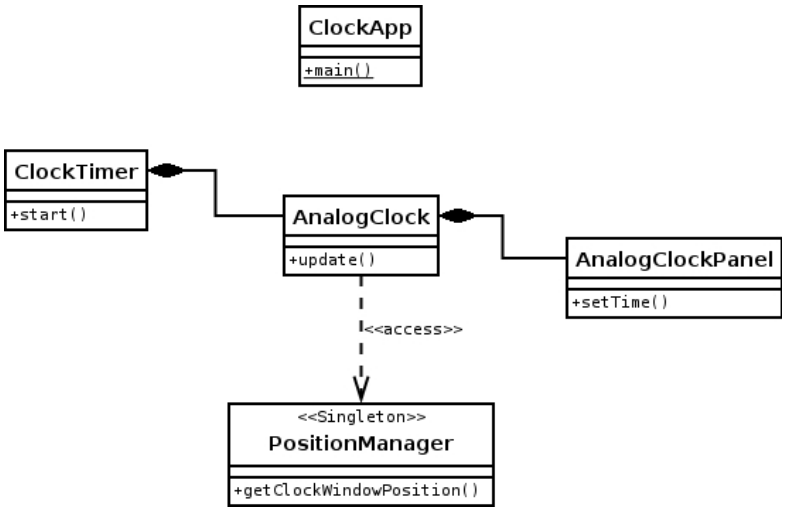

Figure 1: Class diagram of the basic Clock application.

As you can see, the design is far away from an optimal way. It is difficult to extend, difficult to reuse, classes are highly coupled,...

1. Create a new project in your IDE with the classes available online [3].
2. Test the created project and get familiar with the source-code.
3. Improve the desing of the application:
   - Modify the structure of the classes in order to implement the Observer pattern [2]. You are free to rely on `java.util.Observer` and `java.util.Observable` (= Subject) classes available in Java or to implement your own Interface/Classe. Implement the pattern as discussed in the lecture video.
   - Modify the main class in order to be able to add and remove observers dynamically.

- Based on your new design, implement a digital clock.
- Optional: create yet another cool clock. If you lack inspiration, have a look at the site of Tokyoflash [1].

4. Document your architecture with the help of a class diagram and a sequence diagram (in UML).
5. Comment and justify the choices you made (pull/push model—see below)

There are two variants to propagate updated values (= the current time in this exercice).

With the push model, the updated value is passed directly as a parameter of `update()`. With the pull model, a reference to the `Subject` is passed in `update()` so that the `Observer` can get the attributes of the `Subject` (e.g. `mySubject.getValue()`. This is illustrated in the (non-UML) Figure 2.

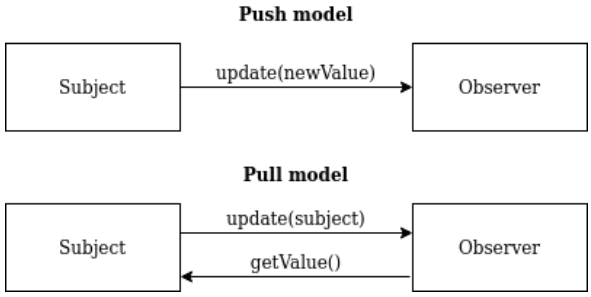

Figure 2: Simplified diagram of the push and pull models.

References

[1] Tokyoflash japan. https://tokyoflash.com/collections/watches (accessed Mar 25, 2021).

[2] Erich GAMMA, Richard HELM, Ralph JOHNSON, and John VLISSIDES. *Design Patterns - Elements of reusable object-oriented software*. Addison-Wesley, 1995.

[3] Jacques Pasquier. Génie logiciel I, 2025. https://moodle.unifr.ch/course/view.php?id=284879 (accessed Mar 19, 2025).