Introduction

In the development of the insulin pump simulator, we focussed on making a modular architecture founded on established software design patterns. We sought to build a system that was maintainable, expandable, and behaved very similarly to a real device. Below is a discussion of how our implementation follows several key design patterns and principles.

---

1. Mediator Pattern (DataManager and NavigationManager as the mediators)

- Every physical or UI element, such as the pump battery, insulin cartridge, CGM sensor, and delivery logic, has its own individual class.

- To make these classes decoupled and manage communication, we introduced mediator-like classes: NavigationManager and DataManager.

- This was good because changes in one element wouldn't impact others directly, increasing modularity and reducing dependencies.

- It organized debugging and testing in a simpler way and allowed scalable feature growth.

---

2. Observer Pattern (Event Handling and Updates)

- We introduced a real-time update system where ten seconds of simulated time passes every hour.

- During activities like active bolus delivery, the system updates the UI and console step by step, displaying the changes both visually and textually.

- Console logging with synchronization to the simulation clock yielded observable output that reflected internal system events like insulin delivery, alarms, and profile change.

- This state-change-based update mechanism allowed observers—like UI components or logs—to react to state changes, making traceability and responsiveness easier.

---

3. State Pattern (Event Handling and Updates)

- The simulation was constructed to simulate different pump states (e.g., idle, bolus delivery) and transition between them as a consequence of user commands or simulated events.

- These states were both visually depicted in the UI and by console outputs, following the rule of clearly depicting runtime behavior.

- State-dependent behavior, like stopping insulin delivery on notifications or resuming on resolution, was encapsulated in a way that transitions were made smooth and deterministic.

---

4. User Interface (HomeScreenWidget as the UI)

- We chose to create the UI manually in code instead of using Qt's drag-and-drop designer because the design tab in Qt is kinda clunky and doesn't display all the available widgets.

- Coding the UI gave precise control over the layout and made it easier to iterate over testing and debugging.

- With the UI logic neatly structured in code, the connection between the user input and system response was more transparent and maintainable.

- This allowed us to more easily understand how UI components linked to backend logic, improving overall system readability.

Conclusion

Our simulation was built with structure, modularity, and real-time feedback in mind. The use of mediator classes, reactive event handling, and clear state transitions ensured that each part of the system operated independently and harmoniously. The handwritten UI creation and logging capabilities also contributed to an interactive and maintainable design, following elementary software engineering principles while effectively simulating the insulin pump.