BSE 2210 Software Design

Assignment 2: Group 2

**Group members**

Tafadzwa Mungandaire 2420974

Kamenga Katendi 2410432

Alintula Silwimba 2410030

**Questions**

Reflect and Document Tradeoffs (20-30 minutes):

In 1-2 paragraphs, explain your choices: Why did you pick those patterns? What tradeoffs

(e.g., simpler design vs. handling more users)? Any basic ethical ideas (e.g., privacy for user

data)?

Note how AI/research helped: "AI suggested Observer, which we adapted because..."

Prepare a quick share: One diagram slide or doc with your models and reflection.

REFLECTION ON DESIGN PATTERNS

**Factory Pattern**

* The Factory pattern could be employed to create different types of notifications (e.g., email, SMS, push notification) without the main business logic knowing the concrete classes.
* When an event status changes (like a new registration or an event becoming full), the Factory produces the appropriate notification object based on user preferences or event context.

**Flexibility:** Easy to add new notification types in the future without modifying existing code.

* **Decoupling:** Keeps notification creation logic separate from registration/event logic.

**Limitations**

* The Factory pattern is excellent for managing notification types but doesn’t manage how or when notifications are sent out in real time to multiple users.

**Singleton Pattern**

* The Singleton pattern could ensure there’s only one instance of the event list or event manager throughout the application.
* It guarantees a single source of truth for event data, preventing conflicting updates from different parts of the system.
* **Consistency:** All users interact with the same instance of the event manager, so event data is always up-to-date.
* **Centralization:** Makes it easier to coordinate updates and logging.

**Limitations**

* Singleton solves the data consistency problem, but doesn’t by itself address notifying multiple clients of changes or handling real-time updates.

**Observer Pattern (Ultimate Choice)**

**For real time notifications we ultimately resolved to Observer pattern here is how it works**

* The Observer pattern allows users (observers) to subscribe to events (subjects). When an event’s state changes (like a registration or event becoming full), all subscribed users are notified instantly.
* In code, this means when the Event object changes, it automatically notifies all registered observers (user interfaces, notification services) so they update in real-time
* **Real-Time Communication:** Instantly propagates state changes to many users, which is essential for this app’s real-time requirements.
* **Decoupling:** The event logic doesn’t need to know about all the users or notification systems; it simply notifies observers.
* **Scalability:** New observers (e.g., admin dashboards, mobile apps) can be added easily.

**Limitations**

* Requires careful management of observer subscriptions to avoid memory leaks or redundant notifications.
* Still needs concurrency control (e.g., with Singleton or database transactions) for simultaneous registrations, but excels at distributing updates.

**Why Observer is the Ultimate Choice**

While the **Factory** and **Singleton** patterns both play important supporting roles (managing notification creation and ensuring a consistent event list), the **Observer pattern directly matches the app’s core need**: real-time, multi-user notification of event changes. It ensures that when one user registers or an event fills up, all interested users are instantly and reliably informed, which is the heart of the system’s value for a community center

How AI Research helped

Helped us understand how the different patterns came to be, that is how they solved challenges faced in the software development and design fields

Analysing the benefits and tradeoffs of the various design patterns