Part1

Wednesday, November 5, 2025 7:52 PM

let divide this into two parts,
1) smarter subscriptions, RAII + weak_ptr + safe snapshot,
2) message filtering
let's scope **Part 1: Smarter Subscriptions** (no filters yet).
You'll build a rock-solid ChatRoom with **RAII tokens, weak_ptr storage, and safe snapshots**. No code below—just the exact requirements you'll implement.

Part 1 — Requirements: RAII + weak_ptr + Safe Snapshot

1) Roles & Interfaces

- **Subject:** ChatRoom
- **Observer:** IChatObserver

IChatObserver

- Method: void update(const std::string& sender, const std::string& text)
- No other responsibilities.

IChatSubject

- Method: Subscription subscribe(const std::shared_ptr<IChatObserver>& obs)
  - Returns a **move-only RAII token**. When the token is destroyed (or cancel() is called) the observer is unsubscribed.
- Utility (optional but recommended):
  - void sendMessage(const std::string& sender, const std::string& text)
  - void removeAll() (unsubscribe all)

2) RAII Subscription Token

- **Move-only** class with:
  - Subscription(std::function<void()> cancel)
  - void cancel() (idempotent)
  - Destructor auto-calls cancel() if still active.
- No copies allowed (delete copy ctor/assign). Move ctor/assign enabled.

3) ChatRoom Storage & Unsubscribe

- Store subscribers as **weak references** to avoid ownership cycles:

  ```
  struct Slot { std::weak_ptr<IChatObserver> obs; };
  std::unordered_map<std::size_t, Slot> subs_;
  std::size_t nextId_ = 1;
  ```
- subscribe():
  - Generate a unique id.
  - Insert {id -> weak_ptr(obs)}.
  - Return a Subscription capturing id and a weak_ptr<ChatRoom> that calls unsubscribe(id) safely.
- unsubscribe(id) is private to ChatRoom.

4) Safe Snapshot Notification

- sendMessage(sender, text) must:
  1. Lock a mutex.
  2. Build a **snapshot vector** of shared_ptr<IChatObserver> by locking each weak_ptr.
  3. **Prune expired** slots from subs_.
  4. Unlock the mutex.
  5. Iterate the snapshot and call update(sender, text) **outside the lock**.
- This allows:
  - Observers to **self-unsubscribe** inside update() (no iterator invalidation).
  - Minimal lock hold time.

5) Thread-Readiness (still single-threaded)

- Add std::mutex m_ and guard:
    - subscribe(), unsubscribe(), removeAll()
    - Snapshot creation & pruning in sendMessage()
- Do **not** spawn threads here; just make access safe if callers later use threads.

6) Observer Examples (simple)

- UserDisplay(name): prints "[name] <- sender: text".
- OnceKeywordBell(name, keyword) (optional for testing self-detach **without filters**):
    - On first message that **contains** keyword, prints a note and **cancels its own Subscription** (you'll inject/store its token).
    - This is just to validate the RAII path; full filtering logic comes in Part 2.

7) Lifecycle & Ownership Rules

- Observers are created as std::shared_ptr<UserDisplay>.
- subscribe() is called and the returned Subscription is stored by the caller.
- If the caller drops the Subscription, the observer is removed automatically.
- If the caller **destroys the observer** object (no more shared owners), the weak_ptr expires; ChatRoom prunes it on next send.

8) Edge Cases

- Subscribing the **same observer instance** twice is allowed but should create **two** independent slots (and two tokens). (You can log or disallow duplicates if you prefer—just be consistent.)
- removeAll() clears every slot regardless of token state (tokens become no-ops if later canceled).
- sendMessage() with **no subscribers** is a no-op.

9) Acceptance Test (what your main() must simulate)

1. Create auto room = std::make_shared<ChatRoom>();
2. Create observers alice, bob, carol (shared_ptr<IChatObserver>).
3. Subscription sA = room->subscribe(alice);
   Subscription sB = room->subscribe(bob);
   Subscription sC = room->subscribe(carol);
4. room->sendMessage("Alice", "hello everyone");
   → All three receive.
5. **Self-detach test (optional):** Subscribe a OnceKeywordBell("Bell","hello"), store its token in the object, and ensure it self-cancels on first hit.
6. **Manual cancel test:** sB.cancel(); → Bob should stop receiving.
7. **Observer destroyed test:** Reset carol's shared_ptr; send another message → no crash; carol is pruned.
8. **removeAll()** then send again → nobody receives.

10) Output Expectations

- Messages print once per live subscription.
- After sB.cancel(), Bob prints nothing further.
- After carol is destroyed, she prints nothing further and no errors.
- After removeAll(), silence.