ChatRoom

Wednesday, November 5, 2025 7:52 PM

💬 Example 3 — Chat Room Broadcaster

Concept:

One **ChatRoom** (Subject) has many **Users** (Observers).
Whenever any user sends a message → the room broadcasts it to *all connected users*.

🧩 Functional Requirements (what to build)

1️⃣ System Overview

- A **ChatRoom** acts as the **Subject**.
- Multiple **User** objects act as **Observers**.
- When a user sends a message, it's sent to the room → which **notifies all users** (except the sender).

2️⃣ Observer Relationship

- Each User **registers** (attach) to the room when they join.
- They **unregister** (detach) when they leave.
- The room can **notify all observers** whenever a message arrives.

3️⃣ Data Flow

1. User.sendMessage("Hello")
2. → ChatRoom receives message and sender reference.
3. → ChatRoom.notifyAll(sender, message)
4. → All users except sender receive: "senderName: Hello"

⚙️ Behavioral Rules

Rule	**Description**
R1	A User has a name (e.g., "Alice")
R2	Each user can join(ChatRoom*) and leave()
R3	When a user sends a message, the ChatRoom distributes it to all *other* users.
R4	The ChatRoom should support attaching/detaching users dynamically.
R5	If no users are in the chat, messages are ignored safely.
R6	The same user cannot be added twice (duplicate-attach guard).
R7	The ChatRoom can clear all users with removeAll() when the room closes.

🧠 Optional Stretch (for later)

#	**Feature**	**Description**
1️⃣	Private Messages	sendPrivate("Bob", "Hi") sends to one observer only
2️⃣	Message Filter	Users can ignore messages containing certain keywords
3️⃣	Logger Observer	A special observer that logs all chat messages to console
4️⃣	AutoLeave Observer	User leaves automatically if message contains "bye"
5️⃣	Multi-room support	One user can observe multiple chat rooms

🔧 Design Hints (use your brainpower 💪)

- Think how your **DoorSensor** and **WeatherStation** patterns apply here.

- Identify:
    - The **Subject interface** — what methods it should expose (attach, detach, notify, broadcastMessage, etc.)
    - The **Observer interface** — what each user must implement (receiveMessage(sender, text) maybe?).
- You'll need to decide:
    - What parameters notify() should have.
    - How to skip the sender during notification.
    - How to model "user leaves chat" safely.

🧩 Challenge Steps for You

Step 1: Define your two interfaces:

- IChatObserver (for users)
- IChatSubject (for the chat room)

Step 2: Implement a simple **ChatRoom** class (as Subject):

- keeps a list/vector of users
- sends messages to them via notifyAll(sender, message)

Step 3: Implement a **User** class (as Observer):

- has a name
- can join(ChatRoom&)
- can sendMessage(ChatRoom&, const std::string&)
- receives messages from others

Step 4: Write main() to simulate:

- create one ChatRoom
- create three users (Alice, Bob, Carol)
- have them join
- Alice sends a message
- Bob replies
- Carol leaves mid-conversation
- Then one more message happens

Step 5 (after you test):

- Add one stretch feature of your choice (private msg, filter, logger, etc.)