Observer Pattern

Thursday, October 30, 2025 8:05 PM

OBSERVER PATTERN LEARNING ROADMAP (C++)

Goal: Go from beginner → intermediate → project-ready mastery in C++.

one simple real-life system where *one source* of truth (a “subject”) informs *many dependents* (“observers”) automatically. (E.g., “A YouTube channel notifying subscribers when a new video is uploaded.”)

Observer Pattern definition (in your own words):

Defines a one-to-many dependency between objects so that when one object (Subject) changes state, all its dependents (Observers) are notified and updated automatically.

✅ Summary — Key Topics to Master

1. ✔️ Basic structure (you already did)
2. 🧠 Push vs Pull models
3. 🧵 Thread-safe implementation
4. 🔄 Dynamic subscription/unsubscription
5. 💡 Smart pointers (weak_ptr / shared_ptr)
6. ⚙️ Template or generic implementation
7. 🚀 Signals/slots (Boost, Qt)
8. 🧩 Event filtering and multi-event subjects
9. 🌍 Real-world applications
10. 🔮 Observer-related advanced patterns (Pub/Sub, Reactive)

🧭 OBSERVER DESIGN PATTERN — COMPLETE C++ COURSE ROADMAP

Total:

- 3 Phases (Beginner → Intermediate → Project)
- **6 Guided Examples**
- **6 Practice Exercises**
- **6 Assignments**
- **5 Mini-Projects**
- Each one builds on the previous stage — no confusion, no leaps.

⚙️ PHASE 1 — BEGINNER (Foundation)

Goal: Learn the pure form of Observer Pattern — one subject, many observers.

🔍 Concepts

1. What is “one-to-many notification”
2. Roles: Subject / Observer
3. attach(), detach(), notify(), update()
4. Push vs Pull model
5. Sequence of calls
6. Avoid coupling between Subject and Observer

🧩 6 GUIDED EXAMPLES

#	**Example**	**Description**
1. Weather Station	Classic example — WeatherData notifies multiple displays (Push model).	
2. Door Sensor	Door opens/closes → multiple alarms/LEDs update.	
3. Chat Room Broadcaster	One chatroom broadcasts messages to all joined users.	
4. Stock Market Feed	StockExchange notifies multiple trader dashboards.	
5. Battery Monitor	Battery level change triggers UI updates.	

6. Event Logger	Logger subject notifies multiple sinks (console/file/network).

Each example introduces one new variation (number of subjects, update style, event type, etc.)

🧠 6 HANDS-ON EXERCISES

#	Exercise	Skill
1	Add another observer type to Weather Station (e.g. ForecastDisplay).	Extend basic structure
2	Detach one observer midway through updates.	Manage lifecycle
3	Implement a "once-only" listener that detaches after first notify.	Control notification count
4	Modify Door Sensor to trigger only on **open → close** transition.	Condition-based event
5	Add timestamps to Chat messages.	Enrich event data
6	Make a simple "Pull" model version of Weather Station.	Compare push vs pull

🚀 PHASE 2 — INTERMEDIATE (Real-world Safety)

Goal: Make your observer systems robust, leak-free, and thread-safe.

🧱 Concepts

1. shared_ptr and weak_ptr lifetimes
2. RAII Subscription (auto detach)
3. Snapshot notify (safe iteration)
4. Filtering (send only selected events)
5. Thread safety (mutex locking)
6. Asynchronous event delivery (queued notifications)

🧩 6 INTERMEDIATE ASSIGNMENTS

#	Assignment	Description
1. Weather Station V2	Add RAII subscription + auto-detach.	
2. Chat Room V2	Support "mute" or "keyword filter" per user.	
3. Stock Market Hub	Use weak_ptr to store observers; auto-remove expired ones.	
4. Battery Monitor (Threaded)	Use a background thread that pushes updates every second.	
5. Logger with Filters	Allow observers to subscribe only to certain log levels.	
6. Event Bus Core	Build a reusable class for topic-based subscriptions.	

🏗️ PHASE 3 — ADVANCED (Mini-Projects)

Goal: Integrate Observer into small but complete applications.

🧩 5 MINI PROJECTS

#	Mini Project
1 **Smart Home Hub**	Multiple sensors (door, light, temperature) → dashboard observers.
2 **Chat Application**	ChatRoom subject → multiple user observers with filters.
3 **Real-Time Stock Dashboard**	Multiple symbols → multiple observers with priorities.
4 **File Download Manager**	Progress updates → UI, logger, and statistics modules.
5 **Game Event System**	Game world publishes events → UI, audio, score modules observe.

Each project will include:

- UML diagram
- Step-by-step coding tasks
- Stretch goals (optional extensions)

🧩 BONUS SECTION — (Phase 4 if you want later)

Observer Composition Patterns

- Observer + State (Reactive states)
- Observer + Command (Event→Action)
- EventBus + Mediator
- Distributed Observer (networked updates)

🧱 COURSE FORMAT

Every Example / Exercise / Assignment follows this consistent flow:

1️⃣ **Problem statement**
2️⃣ **State diagram / concept sketch**
3️⃣ **You code → I review**
4️⃣ **Stretch goal** (optional enhancement)
5️⃣ **Summary of what you learned**