CodeForge - B01 - Strategy Pattern Cơ Bản

Độ khó: ★★ Medium

Đề bài

Tạo Strategy pattern đầu tiên:

- **Strategy** interface `PaymentStrategy` với:
  - `void pay(double amount);`
- **Concrete Strategies**:
  - `CreditCardStrategy`: implement pay() for credit card
  - `PayPalStrategy`: implement pay() for PayPal
  - `CryptoStrategy`: implement pay() for cryptocurrency
- **Context** class `ShoppingCart` với:
  - `PaymentStrategy strategy`
  - `void setStrategy(PaymentStrategy strategy)`
  - `void checkout(double amount)` calls strategy.pay()

Strategy: Family of interchangeable algorithms!

Input

- Dòng 1: Amount
- Dòng 2: Payment method (CREDIT/PAYPAL/CRYPTO)

Output

- Payment processing message

Constraints

- `0 < amount ≤ 1000000`

Ví dụ

Test case 1

Input:

```
1500.00
CREDIT
```

Output:

```
Processing credit card payment: $1500.00
Payment successful via Credit Card
```

Test case 2

Input:

```
500.50
PAYPAL
```

Output:

```
Processing PayPal payment: $500.50
Payment successful via PayPal
```

Tags: strategy, pattern, algorithm, interchangeable, behavioral

CodeForge - B02 - Strategy Eliminates Conditionals

Độ khó: ★★ Medium

Đề bài

Demo Strategy loại bỏ conditionals:

- **Bad approach** (without Strategy):

```
void pay(String type, double amount) {
    if (type.equals("CREDIT")) {
        // credit card logic
    } else if (type.equals("PAYPAL")) {
        // paypal logic
    } else if (type.equals("CRYPTO")) {
        // crypto logic
    }
    // Many if-else!
}
```

- **Good approach** (with Strategy):
  - No conditionals in Context
  - Strategy encapsulates algorithm
  - Easy to add new strategies

◇ Input

- Approach (BAD/GOOD)
- N payments

◇ Output

- Code complexity comparison

◇ Constraints

- $1 \leq N \leq 10$

Ví dụ

Test case 1

Input:

```
BAD
```

Output:

```
Without Strategy Pattern:
- Multiple if-else statements
- Hard to add new payment methods
- Violates Open-Closed Principle
- Context tightly coupled to algorithms
Cyclomatic Complexity: HIGH
```

Test case 2

Input:

```
GOOD
```

Output:

```
With Strategy Pattern:
- No conditionals needed
- Easy to add new strategies
- Follows Open-Closed Principle
- Context loosely coupled
Cyclomatic Complexity: LOW
```

Tags: `strategy`, `conditionals`, `refactoring`, `clean-code`, `pattern`

CodeForge - B03 - Strategy Context Class

Độ khó: ★★ Medium

Đề bài

Hiểu Context trong Strategy pattern:

- **Context** class `Navigator` với:
  - `RouteStrategy strategy`
  - `void setStrategy(RouteStrategy strategy)`
  - `void navigate(String from, String to)`:
    - Delegates to strategy.buildRoute()
- **Strategies**:
  - `CarRouteStrategy`: fastest route
  - `WalkingRouteStrategy`: shortest route
  - `BikeRouteStrategy`: bike-friendly route

Context: Maintains reference to strategy, delegates work

Input

- Dòng 1: From location
- Dòng 2: To location
- Dòng 3: Travel mode (CAR/WALK/BIKE)

Output

- Route details

Constraints

- N/A

Ví dụ

Test case 1

Input:

```
Home
Office
CAR
```

Output:

```
Navigation: Home → Office
Strategy: Car Route
Route: Highway A → Street B → Office (15 min)
Distance: 10 km
Type: Fastest
```

Test case 2

Input:

```
Home
Office
WALK
```

Output:

```
Navigation: Home → Office
Strategy: Walking Route
Route: Park Path → Shortcut → Office (25 min)
Distance: 2 km
Type: Shortest
```

Tags: strategy, context, delegation, pattern

CodeForge - B04 - Runtime Strategy Selection

Độ khó: ★ ★ ★ Hard

Đề bài

Switch strategies at runtime:

- Context: `FileCompressor`
- Strategies:
  - `ZipCompressionStrategy`: .zip format
  - `RarCompressionStrategy`: .rar format
  - `GzipCompressionStrategy`: .gz format
- Select strategy based on file extension or user choice
- Change strategy dynamically during runtime

◇ Input

- Dòng 1: N (files)
- N dòng: Filename, preferred format

◇ Output

- Compression log

◇ Constraints

- `1 ≤ N ≤ 20`

Ví dụ

Test case 1

Input:

```
3
document.txt ZIP
image.jpg RAR
data.csv GZIP
```

Output:

```
File 1: document.txt
[Strategy] Switching to ZIP compression
Compressing with ZIP...
✓ Saved as document.zip (60% compressed)
```

```
File 2: image.jpg
[Strategy] Switching to RAR compression
Compressing with RAR...
✓ Saved as image.rar (40% compressed)

File 3: data.csv
[Strategy] Switching to GZIP compression
Compressing with GZIP...
✓ Saved as data.csv.gz (70% compressed)
```

Tags: strategy, runtime, dynamic, flexible, pattern

CodeForge - B05 - When To Use Strategy Pattern

Độ khó: ★ ★ Medium

Đề bài

Khi nào dùng Strategy:

- ☑ Multiple algorithms for same task
- ☑ Want to switch algorithms at runtime
- ☑ Eliminate long if-else chains
- ☑ Follow Open-Closed Principle
- ❌ Only one algorithm → no need Strategy
- ❌ Algorithms never change → simple method OK

Demo scenarios.

Input

- Scenario (STRATEGY/SIMPLE)

Output

- Recommendation

Constraints

- N/A

Ví dụ

Test case 1

Input:

```
STRATEGY
```

Output:

```
Scenario: Sorting with multiple algorithms
Recommendation: Use Strategy Pattern
Reasons:
- Multiple algorithms (Bubble, Quick, Merge)
- Choose based on data size
- Easy to add new algorithms
```

```
- No if-else chains needed
Example: Sorter.setStrategy(new QuickSort())
```

Test case 2

Input:

```
SIMPLE
```

Output:

```
Scenario: Single fixed calculation
Recommendation: Simple Method
Reasons:
- Only one algorithm
- Never changes
- Strategy overhead unnecessary
Example: double result = calculate(x, y);
```

Tags: strategy, when-to-use, design-decision, pattern

CodeForge - B06 - Observer Pattern Cơ Bản

Độ khó: ★★ Medium

Đề bài

Tạo Observer pattern đầu tiên:

- **Subject** class `NewsAgency` với:
  - `List<Observer> observers`
  - `void attach(Observer o)` add observer
  - `void detach(Observer o)` remove observer
  - `void notifyObservers(String news)` notify all
- **Observer** interface `NewsSubscriber` với:
  - `void update(String news);`
- **Concrete Observers**:
  - `EmailSubscriber`: receive via email
  - `SMSSubscriber`: receive via SMS
  - `AppSubscriber`: receive via app notification

Observer: One-to-many dependency!

Input

- Dòng 1: N (subscribers)
- N dòng: Subscriber type (EMAIL/SMS/APP)
- Dòng N+2: M (news)
- M dòng: News headlines

Output

- Notification log

Constraints

- $1 \leq N \leq 20$
- $1 \leq M \leq 10$

Ví dụ

Test case 1

Input:

```
3
EMAIL
SMS
```

```
APP
2
Breaking_News_Update
Weather_Alert
```

Output:

```
[NewsAgency] Attaching subscribers...
✓ Email subscriber attached
✓ SMS subscriber attached
✓ App subscriber attached

News 1: Breaking_News_Update
Notifying 3 subscribers...
→ [Email] Sending to inbox: Breaking_News_Update
→ [SMS] Sending to phone: Breaking_News_Update
→ [App] Push notification: Breaking_News_Update

News 2: Weather_Alert
Notifying 3 subscribers...
→ [Email] Sending to inbox: Weather_Alert
→ [SMS] Sending to phone: Weather_Alert
→ [App] Push notification: Weather_Alert
```

Tags: observer, pattern, subject, one-to-many, behavioral

CodeForge - B07 - Subject-Observer Relationship

Độ khó: ★★ Medium

Đề bài

Demo Subject-Observer relationship:

- **Subject** (Publisher):
  - Maintains list of observers
  - State changes trigger notifications
  - Doesn't know concrete observer types
- **Observer** (Subscriber):
  - Registers with subject
  - Receives updates automatically
  - Can unsubscribe anytime

Loose coupling: Subject and Observer independent!

Input

- Dòng 1: N (initial observers)
- N dòng: Observer IDs
- Dòng N+2: M (operations)
- M dòng: NOTIFY/ATTACH/DETACH operations

Output

- Operation log

Constraints

- $1 \leq N \leq 10$
- $1 \leq M \leq 20$

Ví dụ

Test case 1

Input:

```
2
Observer1
Observer2
4
NOTIFY State_A
ATTACH Observer3
```

```
NOTIFY State_B
DETACH Observer1
```

Output:

```
Initial observers: 2

Operation 1: NOTIFY State_A
→ Observer1 received: State_A
→ Observer2 received: State_A

Operation 2: ATTACH Observer3
✓ Observer3 attached
Total observers: 3

Operation 3: NOTIFY State_B
→ Observer1 received: State_B
→ Observer2 received: State_B
→ Observer3 received: State_B

Operation 4: DETACH Observer1
✓ Observer1 detached
Total observers: 2
```

Tags: observer, subject, relationship, loose-coupling, pattern

CodeForge - B08 - Push Vs Pull Observer

Độ khó: ★ ★ ★ Hard

Đề bài

So sánh Push vs Pull model:

- **Push model**:
  - Subject pushes data to observers
  - `void update(Data data)`
  - Observers receive all data
- **Pull model**:
  - Subject notifies without data
  - `void update()`
  - Observers pull data they need: `subject.getData()`
  - More flexible

Demo cả 2 approaches.

Input

- Model type (PUSH/PULL)
- State updates

Output

- Update mechanism

Constraints

- N/A

Ví dụ

Test case 1

Input:

```
PUSH
Temperature_25C
```

Output:

```
Push Model:
[Subject] Pushing data to all observers
```

```
→ Observer1: Received Temperature_25C (may not need it)
→ Observer2: Received Temperature_25C (may not need it)
→ Observer3: Received Temperature_25C (may not need it)

Disadvantage: All observers get all data
```

Test case 2

Input:

```
PULL
Temperature_25C
```

Output:

```
Pull Model:
[Subject] Notifying observers (no data pushed)
→ Observer1: Pulling needed data... Temperature only
→ Observer2: Pulling needed data... Temperature + Humidity
→ Observer3: Not pulling (doesn't need data)

Advantage: Observers get only what they need
```

Tags: observer, push, pull, model, comparison, pattern

CodeForge - B09 - Pub-Sub Model

Độ khó: ★ ★ ★ Hard

Đề bài

Observer pattern = Pub-Sub (Publish-Subscribe):

- **Publisher** (Subject):
  - Publishes events/messages
  - Doesn't know subscribers
- **Subscribers** (Observers):
  - Subscribe to topics/events
  - Receive relevant notifications only
- **Event types**:
  - UserRegistered
  - OrderPlaced
  - PaymentReceived

◇ Input

- Dòng 1: N (subscribers)
- N dòng: Subscriber name, interested events
- Dòng N+2: M (events)
- M dòng: Event type, data

◇ Output

- Event notifications

◇ Constraints

- `1 ≤ N ≤ 10`
- `1 ≤ M ≤ 20`

Ví dụ

Test case 1

Input:

```
3
EmailService UserRegistered,PaymentReceived
SMSService OrderPlaced
AnalyticsService UserRegistered,OrderPlaced,PaymentReceived
3
UserRegistered Alice
```

```
OrderPlaced Order123
PaymentReceived Payment456
```

Output:

```
=== Pub-Sub System ===

Subscribers:
- EmailService: [UserRegistered, PaymentReceived]
- SMSService: [OrderPlaced]
- AnalyticsService: [All events]

Event 1: UserRegistered (Alice)
Publishing...
→ EmailService notified
→ AnalyticsService notified

Event 2: OrderPlaced (Order123)
Publishing...
→ SMSService notified
→ AnalyticsService notified

Event 3: PaymentReceived (Payment456)
Publishing...
→ EmailService notified
→ AnalyticsService notified
```

Tags: observer, pub-sub, publish-subscribe, events, pattern

CodeForge - B10 - Observer With State Management

Độ khó: ★ ★ ★ Hard

Đề bài

Observer pattern với state management:

- Subject: `StockMarket` với:
  - `String stockSymbol`
  - `double price`
  - State changes trigger notifications
- Observers:
  - `StockDisplay`: display current price
  - `StockAlert`: alert if price > threshold
  - `StockLogger`: log all changes

Each observer reacts differently to state changes.

Input

- Dòng 1: Stock symbol
- Dòng 2: Initial price
- Dòng 3: Alert threshold
- Dòng 4: N (price updates)
- N dòng: New prices

Output

- Observer reactions

Constraints

- `1 ≤ N ≤ 50`

Ví dụ

Test case 1

Input:

```
AAPL
150.00
160.00
3
155.00
```

```
162.00
158.00
```

Output:

```
Stock: AAPL
Initial: $150.00
Alert threshold: $160.00

Observers attached:
✓ StockDisplay
✓ StockAlert (threshold: $160.00)
✓ StockLogger

Update 1: $155.00
→ [Display] AAPL: $155.00
→ [Logger] Logged: AAPL changed to $155.00

Update 2: $162.00
→ [Display] AAPL: $162.00
→ [Alert] ⚠ AAPL exceeded threshold: $162.00 > $160.00
→ [Logger] Logged: AAPL changed to $162.00

Update 3: $158.00
→ [Display] AAPL: $158.00
→ [Logger] Logged: AAPL changed to $158.00
```

Tags: observer, state, management, reactions, pattern

CodeForge - B11 - When To Use Observer Pattern

Độ khó: ★★ Medium

📝 Đề bài

Khi nào dùng Observer:

- ☑ One-to-many dependency needed
- ☑ Multiple objects need notification
- ☑ Loose coupling desired
- ☑ Event-driven architecture
- ❌ One-to-one → direct call OK
- ❌ Performance critical → observer overhead

Demo scenarios.

◇ Input

- Scenario (OBSERVER/DIRECT)

◇ Output

- Recommendation

◇ Constraints

- N/A

📊 Ví dụ

Test case 1

Input:

```
OBSERVER
```

Output:

```
Scenario: News notification system
Recommendation: Use Observer Pattern
Reasons:
- Multiple subscribers (email, SMS, app)
- Subscribers can join/leave dynamically
- Loose coupling (NewsAgency doesn't know subscribers)
```

```
- Event-driven (notify on news update)
Benefits: Scalable, flexible, maintainable
```

Test case 2

Input:

```
DIRECT
```

Output:

```
Scenario: Single callback function
Recommendation: Direct Method Call
Reasons:
- Only one listener
- Tight coupling acceptable
- Observer overhead unnecessary
Example: button.onClick(() => handleClick())
```

Tags: observer, when-to-use, design-decision, pattern

CodeForge - B12A - Complete Strategy System - Sorting Algorithms

Độ khó: ★ ★ ★ Hard (Advanced)

Đề bài

Tạo sorting framework với Strategy:

- **Strategy** interface `SortStrategy` với:
  - `void sort(int[] array);`
  - `String getName();`
  - `String getComplexity();`
- **Concrete Strategies**:
  - `BubbleSortStrategy`: O(n^2)
  - `QuickSortStrategy`: O(n log n)
  - `MergeSortStrategy`: O(n log n)
  - `InsertionSortStrategy`: O(n^2)
  - `HeapSortStrategy`: O(n log n)
- **Context** `SortingManager` với:
  - Auto-select strategy based on array size:
    - Small (< 10): Insertion sort
    - Medium (10-1000): Quick sort
    - Large (> 1000): Merge sort
  - Manual strategy selection
  - Performance tracking
- **Benchmark** functionality:
  - Compare all strategies
  - Measure execution time
  - Display complexity

Trong main():

1. Test với different array sizes
2. Auto-select optimal strategy
3. Run benchmarks
4. Compare performance

◇ Input

- Dòng 1: N (test cases)
- N nhóm: Array size, strategy choice (AUTO/MANUAL), data

◇ Output

- Sorting results, performance metrics

◇ Constraints

- 1 ≤ N ≤ 5
- 1 ≤ array size ≤ 10000

📊 Ví dụ

Test case 1

Input:

```
3
5 AUTO 5,2,8,1,9
100 AUTO random
1000 MANUAL MERGE
```

Output:

```
=== Sorting Strategy System ===

Test Case 1: Small Array (size: 5)
Data: [5, 2, 8, 1, 9]
Strategy Selection: AUTO
→ Selected: Insertion Sort (optimal for small arrays)
Complexity: O(n²)

Sorting...
✓ Result: [1, 2, 5, 8, 9]
Time: 0.02ms

---

Test Case 2: Medium Array (size: 100)
Data: [random 100 elements]
Strategy Selection: AUTO
→ Selected: Quick Sort (optimal for medium arrays)
Complexity: O(n log n) average

Sorting...
✓ Result: [sorted array]
Time: 0.5ms

---

Test Case 3: Large Array (size: 1000)
Data: [random 1000 elements]
Strategy Selection: MANUAL (Merge Sort)
→ Selected: Merge Sort
Complexity: O(n log n) worst case
```

```
Sorting...
✓ Result: [sorted array]
Time: 2.1ms

---

=== Benchmark: All Strategies (1000 elements) ===

Bubble Sort:
  Time: 15.2ms
  Complexity: O(n²)
  Status: ❌ Too slow for large data

Insertion Sort:
  Time: 12.8ms
  Complexity: O(n²)
  Status: ❌ Too slow for large data

Quick Sort:
  Time: 2.3ms
  Complexity: O(n log n)
  Status: ✓ Good performance

Merge Sort:
  Time: 2.1ms
  Complexity: O(n log n)
  Status: ✓ Best for large data

Heap Sort:
  Time: 2.5ms
  Complexity: O(n log n)
  Status: ✓ Consistent performance

=== Strategy Pattern Benefits ===
✓ Easy to switch algorithms
✓ Auto-select optimal strategy
✓ Add new algorithms without changing Context
✓ Clean, maintainable code
✓ No if-else chains

Fastest: Merge Sort (2.1ms)
Recommended: Quick Sort (general purpose)
```

Tags: strategy, sorting, algorithms, benchmark, performance, advanced

CodeForge - B13A - Complete Observer System - Weather Station

Độ khó: ★ ★ ★ Hard (Advanced)

Đề bài

Tạo weather monitoring system:

- **Subject** `WeatherStation` với:
  - `float temperature, humidity, pressure`
  - State measurement methods
  - Observer management (attach/detach/notify)
- **Observer** interface `WeatherDisplay` với:
  - `void update(float temp, float humidity, float pressure);`
- **Concrete Observers**:
  - `CurrentConditionsDisplay`: Show current values
  - `StatisticsDisplay`: Track min/max/avg
  - `ForecastDisplay`: Predict based on pressure trend
  - `HeatIndexDisplay`: Calculate heat index
  - `AlertDisplay`: Alert if extreme conditions
- **Display Manager**:
  - Register/unregister displays
  - Broadcast updates

Trong main():

1. Setup weather station
2. Register multiple displays
3. Simulate weather changes
4. Displays update automatically
5. Unregister some displays
6. Continue updates

◇ Input

- Dòng 1: N (displays)
- N dòng: Display types
- Dòng N+2: M (weather updates)
- M dòng: Temp, humidity, pressure

◇ Output

- Display updates

◇ Constraints

- $1 \leq N \leq 10$
- $1 \leq M \leq 50$

Ví dụ

Test case 1

Input:

```
5
CURRENT
STATISTICS
FORECAST
HEATINDEX
ALERT
3
25.5 65.0 1013.2
30.0 70.0 1010.5
35.0 80.0 1008.0
```

Output:

```
=== Weather Monitoring System (Observer Pattern) ===

Registering displays...
✓ Current Conditions Display attached
✓ Statistics Display attached
✓ Forecast Display attached
✓ Heat Index Display attached
✓ Alert Display attached

Total observers: 5

---

Weather Update 1:
Temperature: 25.5°C
Humidity: 65.0%
Pressure: 1013.2 hPa

Broadcasting to 5 displays...

[Current Conditions]
  Temperature: 25.5°C
  Humidity: 65.0%
  Pressure: 1013.2 hPa

[Statistics]
  Avg Temp: 25.5°C
  Min Temp: 25.5°C
```

```
  Max Temp: 25.5°C

[Forecast]
  Pressure trend: Stable
  Prediction: Clear skies

[Heat Index]
  Feels like: 26.2°C

[Alert]
  ✓ All conditions normal

---

Weather Update 2:
Temperature: 30.0°C
Humidity: 70.0%
Pressure: 1010.5 hPa

Broadcasting to 5 displays...

[Current Conditions]
  Temperature: 30.0°C
  Humidity: 70.0%
  Pressure: 1010.5 hPa

[Statistics]
  Avg Temp: 27.8°C
  Min Temp: 25.5°C
  Max Temp: 30.0°C

[Forecast]
  Pressure trend: Falling
  Prediction: Rain possible

[Heat Index]
  Feels like: 33.5°C

[Alert]
  ⚠ Heat index high: 33.5°C

---

Weather Update 3:
Temperature: 35.0°C
Humidity: 80.0%
Pressure: 1008.0 hPa

Broadcasting to 5 displays...

[Current Conditions]
  Temperature: 35.0°C
  Humidity: 80.0%
  Pressure: 1008.0 hPa
```

```
[Statistics]
  Avg Temp: 30.2°C
  Min Temp: 25.5°C
  Max Temp: 35.0°C

[Forecast]
  Pressure trend: Falling rapidly
  Prediction: Storm approaching

[Heat Index]
  Feels like: 42.8°C

[Alert]
  🚨 EXTREME: Temperature 35°C+
  🚨 EXTREME: Heat index 42.8°C (Dangerous!)

---

=== Observer Pattern Benefits ===
✓ Loose coupling (Station doesn't know displays)
✓ Dynamic subscription (add/remove displays)
✓ One-to-many broadcast
✓ Each display reacts independently
✓ Easy to add new display types

Total updates: 3
Total notifications: 15 (5 displays × 3 updates)
```

Tags: observer, weather, monitoring, real-time, pub-sub, advanced

CodeForge - B14A - Complete Strategy System - Game AI

Độ khó: ★ ★ ★ Hard (Advanced)

Đề bài

Tạo game AI với Strategy:

- **Strategy** interface `AIStrategy` với:
  - `Action decide(GameState state);`
  - `String getName();`
  - `String getDifficulty();`
- **Concrete Strategies**:
  - `AggressiveAI`: Attack-focused (Hard)
  - `DefensiveAI`: Defense-focused (Medium)
  - `BalancedAI`: Mix of both (Medium)
  - `RandomAI`: Random moves (Easy)
  - `AdaptiveAI`: Changes based on game state (Expert)
- **Context** `GameCharacter` với:
  - Current strategy
  - Health, position, inventory
  - Switch strategy based on conditions:
    - Low health → Defensive
    - High health → Aggressive
    - Boss fight → Adaptive
- **Game Engine**:
  - Simulate combat
  - Track AI decisions
  - Measure effectiveness

Trong main():

1. Create AI opponents với different strategies
2. Simulate N turns
3. AI adapts to game state
4. Compare strategy effectiveness

◇ Input

- Dòng 1: N (AI characters)
- N dòng: Character name, initial strategy
- Dòng N+2: M (game turns)

◇ Output

- AI decisions, strategy switches

◇ Constraints

- $1 \leq N \leq 5$
- $1 \leq M \leq 20$

📊 Ví dụ

Test case 1

Input:

```
3
Warrior AGGRESSIVE
Mage BALANCED
Assassin ADAPTIVE
10
```

Output:

```
=== Game AI Strategy System ===

Creating AI characters...

Character 1: Warrior
  Strategy: Aggressive AI (Hard)
  Health: 100
  Position: (10, 20)

Character 2: Mage
  Strategy: Balanced AI (Medium)
  Health: 80
  Position: (15, 25)

Character 3: Assassin
  Strategy: Adaptive AI (Expert)
  Health: 90
  Position: (5, 30)

---

Turn 1:
[Warrior - Aggressive AI]
  State: Health 100/100
  Decision: CHARGE_ATTACK
  Target: Nearest enemy
  → Dealt 25 damage

[Mage - Balanced AI]
```

```
  State: Health 80/80
  Decision: CAST_SPELL + MAINTAIN_DISTANCE
  → Dealt 20 damage, moved back

[Assassin - Adaptive AI]
  State: Health 90/90, Enemy count: 3
  Adapting: Many enemies detected
  → Switched to DEFENSIVE mode temporarily
  Decision: STEALTH + WAIT
  → Hidden

---

Turn 5:
[Warrior - Aggressive AI]
  State: Health 45/100
  ⚠ Low health detected!
  → Strategy switch: Aggressive → Defensive
  Decision: BLOCK + HEAL
  → Gained 15 health

[Mage - Balanced AI]
  State: Health 60/80
  Decision: RANGED_ATTACK + MAINTAIN_DISTANCE
  → Dealt 18 damage

[Assassin - Adaptive AI]
  State: Health 70/90, Boss appeared!
  Adapting: Boss fight detected
  → Optimizing for boss pattern
  Decision: EVADE + CRITICAL_STRIKE
  → Dealt 40 damage (critical!)

---

Turn 10:
[Warrior - Defensive AI]
  State: Health 75/100 (recovered)
  Health stabilized
  → Strategy switch: Defensive → Balanced
  Decision: ATTACK + GUARD
  → Dealt 15 damage, blocked 10

[Mage - Balanced AI]
  State: Health 55/80, Mana low
  Decision: CONSERVE_MANA + BASIC_ATTACK
  → Dealt 8 damage

[Assassin - Adaptive AI]
  State: Health 85/90, Boss defeated
  Adapting: Normal enemies remain
  → Switched back to AGGRESSIVE mode
  Decision: BACKSTAB
  → Dealt 35 damage
```

```
---

=== Strategy Effectiveness Analysis ===

Warrior:
  Strategies used: Aggressive → Defensive → Balanced
  Total damage: 250
  Survival rate: 75%
  Adaptability: Medium

Mage:
  Strategy: Balanced (consistent)
  Total damage: 220
  Survival rate: 68%
  Adaptability: Low

Assassin:
  Strategy: Adaptive (dynamic)
  Total damage: 380
  Survival rate: 94%
  Adaptability: High ⭐

=== Strategy Pattern Benefits ===
✓ Dynamic AI behavior
✓ Switch strategies based on game state
✓ Easy to add new AI types
✓ Reusable strategies across characters
✓ Testable AI logic

Best performer: Assassin (Adaptive AI)
Most damage: Assassin (380)
Best survival: Assassin (94%)
```

Tags: strategy, game, ai, adaptive, dynamic, advanced

CodeForge - B15A - Complete System - Event-Driven Architecture

Độ khó: ★ ★ ★ Hard (Advanced)

Đề bài

Tạo complete event system combining Strategy + Observer:

- **Event types** (Strategy pattern):
  - `UserEvent`: Registration, Login, Logout
  - `OrderEvent`: Created, Paid, Shipped, Delivered
  - `SystemEvent`: Error, Warning, Info
- **Event handlers** (Observer pattern):
  - `EmailService`: Subscribe to UserEvent, OrderEvent
  - `SMSService`: Subscribe to OrderEvent
  - `LoggingService`: Subscribe to all events
  - `AnalyticsService`: Subscribe to UserEvent, OrderEvent
  - `NotificationService`: Subscribe to SystemEvent
- **Event bus** (Mediator):
  - Register handlers for event types
  - Publish events
  - Route to appropriate handlers
- **Event processing strategies**:
  - `SynchronousStrategy`: Process immediately
  - `AsynchronousStrategy`: Queue and process later
  - `PriorityStrategy`: High priority first
- **Event Manager** với:
  - Select processing strategy based on event type
  - Manage subscriptions
  - Track event metrics

Trong main():

1. Setup event system
2. Register handlers
3. Publish N events
4. Handlers react based on subscriptions
5. Switch processing strategies
6. Display metrics

◇ Input

- Dòng 1: N (handlers)
- N dòng: Handler type, subscribed events
- Dòng N+2: M (events)

- M dòng: Event type, data, priority

◇ Output

- Event processing log, metrics

◇ Constraints

- $1 \leq N \leq 10$
- $1 \leq M \leq 50$

📊 Ví dụ

Test case 1

Input:

```
5
EmailService USER,ORDER
SMSService ORDER
LoggingService ALL
AnalyticsService USER,ORDER
NotificationService SYSTEM
8
USER Registration Alice NORMAL
USER Login Alice NORMAL
ORDER Created Order123 HIGH
ORDER Paid Order123 HIGH
SYSTEM Error Database_Connection CRITICAL
ORDER Shipped Order123 NORMAL
USER Logout Alice NORMAL
SYSTEM Warning High_CPU NORMAL
```

Output:

```
=== Event-Driven Architecture (Strategy + Observer) ===

Registering event handlers...
✓ EmailService: [USER, ORDER]
✓ SMSService: [ORDER]
✓ LoggingService: [ALL]
✓ AnalyticsService: [USER, ORDER]
✓ NotificationService: [SYSTEM]

Total handlers: 5

Setting processing strategies...
→ USER events: Asynchronous (queue)
→ ORDER events: Priority-based (high → normal)
→ SYSTEM events: Synchronous (immediate)
```

```
---

Event 1: USER Registration (Alice) [NORMAL]
Strategy: Asynchronous
→ Queued for processing
Processing...
  [EmailService] Sending welcome email to Alice
  [LoggingService] Logged: USER Registration Alice
  [AnalyticsService] Tracking: New user Alice

---

Event 2: USER Login (Alice) [NORMAL]
Strategy: Asynchronous
→ Queued for processing
Processing...
  [EmailService] Sending login notification to Alice
  [LoggingService] Logged: USER Login Alice
  [AnalyticsService] Tracking: User login Alice

---

Event 3: ORDER Created (Order123) [HIGH PRIORITY]
Strategy: Priority (HIGH → immediate)
→ Processing immediately
  [EmailService] Order confirmation: Order123
  [SMSService] SMS: Order123 created
  [LoggingService] Logged: ORDER Created Order123
  [AnalyticsService] Tracking: New order Order123

---

Event 4: ORDER Paid (Order123) [HIGH PRIORITY]
Strategy: Priority (HIGH → immediate)
→ Processing immediately
  [EmailService] Payment receipt: Order123
  [SMSService] SMS: Payment confirmed Order123
  [LoggingService] Logged: ORDER Paid Order123
  [AnalyticsService] Tracking: Order paid Order123

---

Event 5: SYSTEM Error (Database_Connection) [CRITICAL]
Strategy: Synchronous (immediate)
→ Processing immediately (blocking)
  [NotificationService] 🚨 CRITICAL: Database_Connection error
  [LoggingService] Logged: SYSTEM Error Database_Connection
⚠️ Processing blocked until handled

---

Event 6: ORDER Shipped (Order123) [NORMAL]
Strategy: Priority (NORMAL → queue)
```

```
→ Queued for processing
Processing...
  [EmailService] Shipping notification: Order123
  [SMSService] SMS: Order123 shipped
  [LoggingService] Logged: ORDER Shipped Order123
  [AnalyticsService] Tracking: Order shipped Order123

---

Event 7: USER Logout (Alice) [NORMAL]
Strategy: Asynchronous
→ Queued for processing
Processing...
  [LoggingService] Logged: USER Logout Alice
  [AnalyticsService] Tracking: User logout Alice

---

Event 8: SYSTEM Warning (High_CPU) [NORMAL]
Strategy: Synchronous (immediate)
→ Processing immediately
  [NotificationService] ⚠ WARNING: High_CPU
  [LoggingService] Logged: SYSTEM Warning High_CPU

---

=== Event Metrics ===

Events Published: 8
  USER: 3
  ORDER: 3
  SYSTEM: 2

Handler Invocations: 28
  EmailService: 6
  SMSService: 3
  LoggingService: 8
  AnalyticsService: 6
  NotificationService: 2

Processing Strategies Used:
  Synchronous: 2 events (immediate)
  Asynchronous: 3 events (queued)
  Priority: 3 events (by priority)

Performance:
  Avg processing time: 12ms
  Synchronous avg: 5ms (fast but blocking)
  Asynchronous avg: 15ms (non-blocking)
  Priority avg: 8ms (balanced)

---

=== Pattern Benefits Demonstrated ===
```

```
✓ Strategy Pattern:
  - Flexible event processing (sync/async/priority)
  - Switch strategies per event type
  - Add new strategies without changing handlers

✓ Observer Pattern:
  - Loose coupling (handlers independent)
  - Dynamic subscriptions
  - One-to-many event broadcast

✓ Combined Power:
  - Scalable event system
  - Flexible and extensible
  - Clean architecture
  - Production-ready design

Total patterns used: 2 (Strategy + Observer)
Architecture: Event-driven, loosely coupled
Code quality: High maintainability
```

Tags: strategy, observer, combined-patterns, event-driven, architecture, capstone, advanced