CodeForge - B01 - Adapter Pattern Cơ Bản

Độ khó: ★★ Medium

📝 Đề bài

Tạo Object Adapter đầu tiên:

- **Target** interface `MediaPlayer` với:
  - `void play(String filename);`
- **Adaptee** class `AdvancedMediaPlayer` với:
  - `void playVLC(String filename)` in "Playing VLC: [filename]"
  - `void playMP4(String filename)` in "Playing MP4: [filename]"
- **Adapter** class `MediaAdapter` implements MediaPlayer với:
  - `AdvancedMediaPlayer advancedPlayer` (composition)
  - `void play(String filename)`:
    - Detect type from filename
    - Call appropriate advancedPlayer method

Adapter: Convert incompatible interface!

◇ Input

- Dòng 1: N (files)
- N dòng: Filename (e.g., song.vlc, video.mp4)

◇ Output

- Playing messages

◇ Constraints

- `1 ≤ N ≤ 20`

📊 Ví dụ

Test case 1

Input:

```
3
song.vlc
video.mp4
music.vlc
```

Output:

```
Playing VLC: song.vlc
Playing MP4: video.mp4
Playing VLC: music.vlc
```

Tags: `adapter`, `object-adapter`, `target`, `adaptee`, `pattern`

CodeForge - B02 - Target-Adaptee-Adapter Components

Độ khó: ★★ Medium

📝 Đề bài

Demo 3 components của Adapter pattern:

- **Target** interface `Socket` với:
    - `Volt get120Volt();`
- **Adaptee** class `EuroSocket` với:
    - `Volt get240Volt()` return 240V
- **Adapter** class `SocketAdapter` implements Socket với:
    - `EuroSocket euroSocket`
    - `Volt get120Volt()`:
        - Get 240V from EuroSocket
        - Convert to 120V
        - Return 120V

Adapter converts voltage!

◇ Input

- Dòng 1: N (devices cần 120V)

◇ Output

- Voltage supplied

◇ Constraints

- `1 ≤ N ≤ 10`

📊 Ví dụ

Test case 1

Input:

```
3
```

Output:

```
```

```
Device 1: Received 120V (adapted from 240V)
Device 2: Received 120V (adapted from 240V)
Device 3: Received 120V (adapted from 240V)
```

Tags: `adapter`, `target`, `adaptee`, `components`, `pattern`

CodeForge - B03 - Object Adapter Vs Class Adapter

Độ khó: ⭐⭐ Medium

📝 Đề bài

So sánh Object Adapter (preferred) vs Class Adapter:

- **Object Adapter** (composition):
  - Uses HAS-A relationship
  - More flexible
  - Can adapt multiple adaptees
  - **Preferred approach**
- **Class Adapter** (inheritance):
  - Uses IS-A relationship
  - Less flexible
  - Single inheritance limitation (Java)
  - Not recommended

Demo Object Adapter implementation.

◇ Input

- Adapter type (OBJECT/CLASS)

◇ Output

- Characteristics

◇ Constraints

- N/A

📊 Ví dụ

Test case 1

Input:

```
OBJECT
```

Output:

```
Object Adapter (Composition)
Flexibility: HIGH
Multiple adaptees: YES
```

```
  Maintainability: GOOD
  Coupling: LOW
  Recommended: YES
```

Tags: `adapter`, `object-adapter`, `class-adapter`, `comparison`

CodeForge - B04 - Real-World Adapter - Payment Gateway

Độ khó: ★★★ Hard

📝 Đề bài

Adapter cho payment gateways:

- **Target** interface `PaymentProcessor` với:
  - `boolean processPayment(double amount);`
- **Adaptees** (third-party APIs):
  - `PayPalAPI` với `void sendPayment(double dollars)`
  - `StripeAPI` với `boolean charge(int cents)`
- **Adapters**:
  - `PayPalAdapter` implements PaymentProcessor
  - `StripeAdapter` implements PaymentProcessor
  - Convert methods và units

◇ Input

- Dòng 1: N (payments)
- N dòng: Gateway type, amount

◇ Output

- Payment results

◇ Constraints

- `1 ≤ N ≤ 50`

📊 Ví dụ

Test case 1

Input:

```
3
PAYPAL 100.50
STRIPE 50.75
PAYPAL 200.00
```

Output:

```
[PayPal Adapter] Processing $100.50
  → PayPal API: sendPayment($100.50)
  ✓ Payment successful

[Stripe Adapter] Processing $50.75
  → Stripe API: charge(5075 cents)
  ✓ Payment successful

[PayPal Adapter] Processing $200.00
  → PayPal API: sendPayment($200.00)
  ✓ Payment successful
```

Tags: `adapter`, `real-world`, `payment`, `third-party-api`, `pattern`

CodeForge - B05 - When To Use Adapter Pattern

Độ khó: ⭐⭐ Medium

📝 Đề bài

Khi nào dùng Adapter:

- ☑ Integrate legacy code with new system
- ☑ Use third-party library with incompatible interface
- ☑ Want to reuse existing class
- ☑ Need to create reusable class
- ❌ Can modify existing code → directly fix
- ❌ Simple conversion → utility method OK

Demo scenarios.

◇ Input

- Scenario (ADAPTER/DIRECT)

◇ Output

- Recommendation

◇ Constraints

- N/A

📊 Ví dụ

Test case 1

Input:

```
ADAPTER
```

Output:

```
Scenario: Integrating third-party logging library
Recommendation: Use Adapter Pattern
Reasons:
- Cannot modify third-party code
- Incompatible interface
- Need standard interface for app
- Future library changes isolated
```

Test case 2

Input:

```
DIRECT
```

Output:

```
Scenario: Simple data format conversion
Recommendation: Use Utility Method
Reasons:
- Simple conversion logic
- No interface incompatibility
- Adapter overhead unnecessary
- Direct approach cleaner
```

Tags: `adapter`, `when-to-use`, `design-decision`, `pattern`

CodeForge - B06 - Decorator Pattern Cơ Bản

Độ khó: ★★ Medium

📝 Đề bài

Tạo Decorator đầu tiên:

- **Component** interface `Coffee` với:
  - `String getDescription();`
  - `double getCost();`
- **Concrete Component** class `SimpleCoffee` implements Coffee:
  - Description: "Simple Coffee"
  - Cost: 5.0
- **Decorator** abstract class `CoffeeDecorator` implements Coffee:
  - `protected Coffee coffee` (wraps component)
  - Delegates to wrapped coffee
- **Concrete Decorators**:
  - `MilkDecorator`: +$1.0, "with Milk"
  - `SugarDecorator`: +$0.5, "with Sugar"

Decorator: Add functionality dynamically!

◇ Input

- Dòng 1: N (decorators)
- N dòng: Decorator type (MILK/SUGAR)

◇ Output

- Final description and cost

◇ Constraints

- `0 ≤ N ≤ 10`

📊 Ví dụ

Test case 1

Input:

```
2
MILK
SUGAR
```

Output:

```
Simple Coffee with Milk with Sugar
Cost: $6.50
```

Test case 2

Input:

```
0
```

Output:

```
Simple Coffee
Cost: $5.00
```

Tags: `decorator`, `component`, `wrapper`, `dynamic`, `pattern`

CodeForge - B07 - Decorator Stacking (Chaining)

Độ khó: ★★ Medium

📝 Đề bài

Decorator có thể stack nhiều lớp:

- Component: `Pizza` (base $10)
- Decorators:
  - `CheeseDecorator`: +$2
  - `OlivesDecorator`: +$1.5
  - `MushroomsDecorator`: +$2.5

Usage:

```
Pizza pizza = new SimplePizza();
pizza = new CheeseDecorator(pizza);
pizza = new OlivesDecorator(pizza);
pizza = new MushroomsDecorator(pizza);
```

Stacking: Wrap decorators around decorators!

◇ Input

- Dòng 1: N (toppings)
- N dòng: Topping types

◇ Output

- Pizza với all toppings and total cost

◇ Constraints

- `1 ≤ N ≤ 10`

📊 Ví dụ

Test case 1

Input:

```
3
CHEESE
OLIVES
MUSHROOMS
```

Output:

```
Simple Pizza + Cheese + Olives + Mushrooms
Total: $16.00
```

Tags: `decorator`, `stacking`, `chaining`, `multiple-wrappers`, `pattern`

CodeForge - B08 - Decorator Vs Subclassing

Độ khó: ⭐⭐⭐ Hard

📝 Đề bài

So sánh Decorator vs Subclassing:

- **Subclassing problem**: Combinatorial explosion!
  - SimpleCoffee
  - CoffeeWithMilk
  - CoffeeWithSugar
  - CoffeeWithMilkAndSugar
  - CoffeeWithMilkAndSugarAndWhippedCream
  - ... 2^N combinations!
- **Decorator solution**: Dynamic composition!
  - SimpleCoffee + decorators
  - Mix and match at runtime
  - N decorators, infinite combinations

Demo cả 2 approaches.

◇ Input

- Approach (SUBCLASS/DECORATOR)
- Decorators needed

◇ Output

- Number of classes needed

◇ Constraints

- N/A

📊 Ví dụ

Test case 1

Input:

```
SUBCLASS 5
```

Output:

```
```

```
Subclassing Approach:
With 5 decorators (Milk, Sugar, Whip, Caramel, Vanilla)
Classes needed: 32 (2^5 combinations)
Maintainability: POOR
Flexibility: LOW
```

Test case 2

Input:

```
DECORATOR 5
```

Output:

```
Decorator Approach:
With 5 decorators
Classes needed: 6 (1 base + 5 decorators)
Maintainability: GOOD
Flexibility: HIGH
Runtime composition: YES
```

Tags: `decorator`, `subclassing`, `comparison`, `alternative`, `pattern`

CodeForge - B09 - Decorator Order Matters

Độ khó: ★★★ Hard

📝 Đề bài

Thứ tự decorators ảnh hưởng kết quả:

- Component: `DataSource` với `void write(String data)`
- Decorators:
  - `EncryptionDecorator`: Encrypt data before write
  - `CompressionDecorator`: Compress data before write

Order 1: Compress → Encrypt → Write **Order 2**: Encrypt → Compress → Write

Different results! Demo cả 2.

◇ Input

- Dòng 1: Data
- Dòng 2: Order (COMPRESS_ENCRYPT hoặc ENCRYPT_COMPRESS)

◇ Output

- Processing steps

◇ Constraints

- Độ dài data ≤ 200

📊 Ví dụ

Test case 1

Input:

```
Hello_World
COMPRESS_ENCRYPT
```

Output:

```
Original: Hello_World
Step 1: Compressing... → HW_compressed
Step 2: Encrypting... → HW_enc_comp
Step 3: Writing to file
[Better: Compress first (smaller), then encrypt]
```

Test case 2

Input:

```
Hello_World
ENCRYPT_COMPRESS
```

Output:

```
Original: Hello_World
Step 1: Encrypting... → HW_encrypted
Step 2: Compressing... → HW_comp_enc
Step 3: Writing to file
[May not compress well (encrypted data random)]
```

Tags: `decorator`, `order`, `sequence`, `behavior`, `pattern`

CodeForge - B10 - Decorator Với Behavior Addition

Độ khó: ★★★ Hard

📝 Đề bài

Decorator add new behaviors (not just data):

- Component: `Window` với `void render();`
- Decorators add functionality:
  - `ScrollbarDecorator`: add scroll behavior
  - `BorderDecorator`: add border rendering
  - `ShadowDecorator`: add shadow effect

Each decorator wraps and enhances behavior.

◇ Input

- Dòng 1: N (decorators)
- N dòng: Decorator types

◇ Output

- Rendering sequence

◇ Constraints

- `1 ≤ N ≤ 5`

📊 Ví dụ

Test case 1

Input:

```
3
SCROLLBAR
BORDER
SHADOW
```

Output:

```
Rendering window:
→ Rendering shadow effect
→ Rendering border
→ Adding scrollbars
```

```
  → Rendering basic window
[Decorators execute in reverse order: Shadow → Border → Scrollbar → Base]
```

Tags: `decorator`, `behavior`, `enhancement`, `functionality`, `pattern`

CodeForge - B11 - When To Use Decorator Pattern

Độ khó: ⭐⭐ Medium

📝 Đề bài

Khi nào dùng Decorator:

- ✅ Add responsibilities to objects dynamically
- ✅ Extend functionality without subclassing
- ✅ Combine behaviors flexibly
- ✅ Follow Open-Closed Principle
- ❌ Single enhancement → simple subclass OK
- ❌ Need to remove functionality → not suitable

Demo scenarios.

◇ Input

- Scenario (DECORATOR/SUBCLASS)

◇ Output

- Recommendation

◇ Constraints

- N/A

📊 Ví dụ

Test case 1

Input:

```
DECORATOR
```

Output:

```
Scenario: UI components with optional features
Recommendation: Use Decorator Pattern
Reasons:
- Many optional combinations
- Runtime flexibility needed
- Avoid class explosion
- Mix and match features
Example: Window + Scrollbar + Border + Shadow
```

Test case 2

Input:

```
SUBCLASS
```

Output:

```
Scenario: Single fixed enhancement
Recommendation: Use Simple Subclass
Reasons:
- Only one enhancement
- No dynamic composition needed
- Decorator overhead unnecessary
Example: SpecialButton extends Button
```

Tags: `decorator`, `when-to-use`, `design-decision`, `pattern`

CodeForge - B12A - Complete Adapter System - Data Format Converter

Độ khó: ★★★ Hard (Advanced)

📝 Đề bài

Tạo data format conversion system:

- **Target** interface `DataProcessor` với:
  - `void process(String data);`
  - `String getFormat();`
- **Adaptees** (third-party libraries):
  - `XMLProcessor`: `void parseXML(String xml)`
  - `JSONProcessor`: `void parseJSON(String json)`
  - `CSVProcessor`: `void parseCSV(String csv)`
  - `YAMLProcessor`: `void parseYAML(String yaml)`
- **Adapters**:
  - `XMLAdapter`, `JSONAdapter`, `CSVAdapter`, `YAMLAdapter`
  - Each implements DataProcessor
  - Convert data format if needed
- **Client** `DataPipeline` với:
  - `List<DataProcessor> processors`
  - `void addProcessor(DataProcessor processor)`
  - `void execute(String data, String format)`

Trong main():

1. Create adapters for different formats
2. Build processing pipeline
3. Process data through all adapters
4. Handle format conversions

◇ Input

- Dòng 1: Source format
- Dòng 2: Data
- Dòng 3: N (target formats)
- N dòng: Target format types

◇ Output

- Conversion and processing log

◇ Constraints

- `1 ≤ N ≤ 5`

📊 Ví dụ

Test case 1

Input:

```
JSON
{"name":"Alice","age":30}
3
XML
CSV
YAML
```

Output:

```
=== Data Format Adapter System ===

Source: JSON
Data: {"name":"Alice","age":30}

Building processing pipeline...
✓ XML Adapter added
✓ CSV Adapter added
✓ YAML Adapter added

Processing through pipeline:

[JSON → XML Adapter]
  Converting JSON to XML...
  <person>
    <name>Alice</name>
    <age>30</age>
  </person>
  ✓ XML processing complete

[JSON → CSV Adapter]
  Converting JSON to CSV...
  name,age
  Alice,30
  ✓ CSV processing complete

[JSON → YAML Adapter]
  Converting JSON to YAML...
  person:
    name: Alice
    age: 30
  ✓ YAML processing complete
```

```
=== Summary ===
Source Format: JSON
Conversions: 3
All adapters successfully converted data
```

Tags: `adapter`, `data-format`, `conversion`, `pipeline`, `advanced`

CodeForge - B13A - Complete Decorator System - Logging Framework

Độ khó: ★★★ Hard (Advanced)

📝 Đề bài

Tạo flexible logging framework:

- **Component** interface `Logger` với:
  - `void log(String message);`
- **Concrete Component** `BasicLogger`:
  - Simple console output
- **Decorators**:
  - `TimestampDecorator`: Add timestamp
  - `LevelDecorator`: Add log level (INFO/WARN/ERROR)
  - `FileDecorator`: Write to file
  - `ColorDecorator`: Add ANSI colors
  - `EncryptionDecorator`: Encrypt sensitive logs
- **Logger Factory** để build loggers với different decorator combinations

Trong main():

1. Create different logger configurations
2. Stack decorators dynamically
3. Log messages with different setups
4. Compare outputs

◇ Input

- Dòng 1: N (logger configurations)
- N nhóm:
  - Logger name
  - M decorators
  - K messages

◇ Output

- Log outputs với different decorator stacks

◇ Constraints

- `1 ≤ N ≤ 5`
- `1 ≤ M ≤ 5`
- `1 ≤ K ≤ 10`

📊 Ví dụ

Test case 1

Input:

```
3
ConsoleLogger 2 TIMESTAMP LEVEL 2 INFO Application_started ERROR Connection_failed
FileLogger 3 TIMESTAMP LEVEL FILE 1 INFO Data_saved
SecureLogger 4 TIMESTAMP LEVEL FILE ENCRYPTION 1 ERROR Sensitive_data_breach
```

Output:

```
=== Flexible Logging Framework (Decorator Pattern) ===

Configuration 1: ConsoleLogger
Decorators: TIMESTAMP → LEVEL
Messages: 2

[2024-12-22 10:30:00] [INFO] Application_started
[2024-12-22 10:30:01] [ERROR] Connection_failed

---

Configuration 2: FileLogger
Decorators: TIMESTAMP → LEVEL → FILE
Messages: 1

[2024-12-22 10:30:02] [INFO] Data_saved
  → Written to log.txt

---

Configuration 3: SecureLogger
Decorators: TIMESTAMP → LEVEL → FILE → ENCRYPTION
Messages: 1

[2024-12-22 10:30:03] [ERROR] U2VuU2l0aXZlX2RhdGFfYnJlYWNo (encrypted)
  → Written to secure_log.txt (encrypted)

---

=== Decorator Benefits Demonstrated ===
✓ Dynamic composition at runtime
✓ Mix and match features
✓ Single Responsibility (each decorator one job)
✓ Open-Closed Principle (extend without modify)

Total Configurations: 3
Total Decorators Used: 9
Total Messages Logged: 4
All without subclass explosion!
```

Tags: `decorator`, `logging`, `framework`, `flexible`, `stacking`, `advanced`

CodeForge - B14A - Complete Adapter System - Database Abstraction Layer

Độ khó: ★★★ Hard (Advanced)

📝 Đề bài

Tạo database abstraction với adapters:

- **Target** interface `Database` với:
  - `void connect(String url);`
  - `ResultSet executeQuery(String sql);`
  - `int executeUpdate(String sql);`
  - `void disconnect();`
- **Adaptees** (different database drivers):
  - `MySQLDriver`: native MySQL methods
  - `PostgreSQLDriver`: native PostgreSQL methods
  - `MongoDBDriver`: NoSQL methods (very different!)
  - `SQLiteDriver`: embedded DB methods
- **Adapters**:
  - Convert unified interface to specific driver calls
  - Handle SQL → NoSQL conversion for MongoDB
  - Manage connection pooling
- **Repository Pattern** với:
  - Uses Database interface
  - CRUD operations
  - Works with any database via adapters

Trong main():

1. Create different database adapters
2. Switch databases at runtime
3. Execute same operations on different DBs
4. Compare performance and compatibility

◇ Input

- Dòng 1: N (database types to test)
- N nhóm: DB type, operations

◇ Output

- Execution log for each database

◇ Constraints

- `1 ≤ N ≤ 4`

📊 Ví dụ

Test case 1

Input:

```
3
MYSQL INSERT_User_Alice SELECT_User_Alice
POSTGRES INSERT_User_Bob UPDATE_User_Bob_NewName
MONGODB INSERT_Document_Charlie FIND_Document_Charlie
```

Output:

```
=== Database Abstraction Layer (Adapter Pattern) ===

Testing Database 1: MySQL
[MySQL Adapter] Connecting to MySQL...
  → Native: mysql_connect()
  ✓ Connected

[MySQL Adapter] INSERT User: Alice
  SQL: INSERT INTO users (name) VALUES ('Alice')
  → Native: mysql_query()
  ✓ 1 row inserted

[MySQL Adapter] SELECT User: Alice
  SQL: SELECT * FROM users WHERE name = 'Alice'
  → Native: mysql_query()
  ✓ Result: User{id=1, name=Alice}

[MySQL Adapter] Disconnecting...
  ✓ Disconnected

---

Testing Database 2: PostgreSQL
[PostgreSQL Adapter] Connecting to PostgreSQL...
  → Native: pg_connect()
  ✓ Connected

[PostgreSQL Adapter] INSERT User: Bob
  SQL: INSERT INTO users (name) VALUES ('Bob')
  → Native: pg_query()
  ✓ 1 row inserted

[PostgreSQL Adapter] UPDATE User: Bob → NewName
  SQL: UPDATE users SET name = 'NewName' WHERE name = 'Bob'
  → Native: pg_query()
  ✓ 1 row updated

[PostgreSQL Adapter] Disconnecting...
  ✓ Disconnected

---

Testing Database 3: MongoDB
[MongoDB Adapter] Connecting to MongoDB...
  → Native: mongo_connect()
  ✓ Connected

[MongoDB Adapter] INSERT Document: Charlie
  Converting SQL to NoSQL...
  → Native: db.users.insertOne({name: "Charlie"})
  ✓ Document inserted

[MongoDB Adapter] FIND Document: Charlie
  Converting SQL to NoSQL...
  → Native: db.users.findOne({name: "Charlie"})
  ✓ Result: {_id: ObjectId(...), name: "Charlie"}

[MongoDB Adapter] Disconnecting...
  ✓ Disconnected

---

=== Adapter Pattern Benefits ===
✓ Unified interface for different databases
✓ Switch databases without changing business logic
✓ SQL → NoSQL conversion handled by adapter
✓ Legacy code integration seamless

Databases Tested: 3
Total Operations: 6
All successful through adapter interface!
```

Tags: `adapter`, `database`, `abstraction`, `sql`, `nosql`, `advanced`

CodeForge - B15A - Complete System - Stream Processing Pipeline

Độ khó: ★★★ Hard (Advanced)

📝 Đề bài

Tạo complete system combining Adapter + Decorator:

- **Component** interface `DataStream` với:
  - `void write(byte[] data);`
  - `byte[] read();`
- **Adapters** (different data sources):
  - `FileStreamAdapter`: Adapt file I/O
  - `NetworkStreamAdapter`: Adapt network I/O
  - `MemoryStreamAdapter`: Adapt in-memory buffer
- **Decorators** (processing layers):
  - `BufferingDecorator`: Add buffering
  - `CompressionDecorator`: Add compression (GZIP)
  - `EncryptionDecorator`: Add encryption (AES)
  - `ChecksumDecorator`: Add integrity check
  - `LoggingDecorator`: Add operation logging
- **Pipeline Builder** với:
  - Fluent API to construct pipelines
  - `source(Adapter)` → `buffer()` → `compress()` → `encrypt()` → `checksum()` → `log()`
- **Stream Manager** với:
  - Manage multiple pipelines
  - Execute operations
  - Collect statistics

Trong main():

1. Create different pipeline configurations
2. Process data through pipelines
3. Demonstrate adapter + decorator combination
4. Compare performance với/không decorators

◇ Input

- Dòng 1: N (pipelines)
- N nhóm: Source type, decorators, data

◇ Output

- Processing log, performance metrics

◇ Constraints

- `1 ≤ N ≤ 5`

📊 Ví dụ

Test case 1

Input:

```
3
FILE BUFFER COMPRESS ENCRYPT LOG HelloWorld
NETWORK COMPRESS CHECKSUM Buffer12345
MEMORY ENCRYPT LOG SecretData
```

Output:

```
=== Stream Processing Pipeline (Adapter + Decorator) ===

Pipeline 1: File Source
Source: FileStreamAdapter
Decorators: BUFFER → COMPRESS → ENCRYPT → LOG
Data: HelloWorld (10 bytes)

[Processing Pipeline 1]
→ [FileAdapter] Reading from file...
→ [BufferingDecorator] Buffering 10 bytes...
→ [CompressionDecorator] Compressing (GZIP)...
    Original: 10 bytes
    Compressed: 8 bytes (20% reduction)
→ [EncryptionDecorator] Encrypting (AES-256)...
    Encrypted: 16 bytes (padded)
→ [LoggingDecorator] Logging operation...
    [2024-12-22 10:30:00] FILE_WRITE: 16 bytes
✓ Pipeline 1 complete: 10 → 16 bytes (5ms)

---

Pipeline 2: Network Source
Source: NetworkStreamAdapter
Decorators: COMPRESS → CHECKSUM
Data: Buffer12345 (12 bytes)

[Processing Pipeline 2]
→ [NetworkAdapter] Reading from network socket...
→ [CompressionDecorator] Compressing (GZIP)...
    Original: 12 bytes
    Compressed: 9 bytes (25% reduction)
→ [ChecksumDecorator] Calculating checksum...
    Checksum: 0xABCD1234
    Appended: 4 bytes
✓ Pipeline 2 complete: 12 → 13 bytes (3ms)

---

Pipeline 3: Memory Source
Source: MemoryStreamAdapter
Decorators: ENCRYPT → LOG
Data: SecretData (10 bytes)

[Processing Pipeline 3]
→ [MemoryAdapter] Reading from memory buffer...
→ [EncryptionDecorator] Encrypting (AES-256)...
    Encrypted: 16 bytes (padded)
→ [LoggingDecorator] Logging operation...
    [2024-12-22 10:30:01] MEMORY_WRITE: 16 bytes
✓ Pipeline 3 complete: 10 → 16 bytes (2ms)

---

=== Performance Comparison ===

Without Decorators (Raw):
  Pipeline 1: 10 bytes → 10 bytes (1ms)
  Pipeline 2: 12 bytes → 12 bytes (1ms)
  Pipeline 3: 10 bytes → 10 bytes (1ms)
  Total: 3ms

With Decorators:
  Pipeline 1: 10 bytes → 16 bytes (5ms)
  Pipeline 2: 12 bytes → 13 bytes (3ms)
  Pipeline 3: 10 bytes → 16 bytes (2ms)
  Total: 10ms

Overhead: 7ms
But gained: Compression, Encryption, Logging, Checksums!

=== Pattern Benefits ===

✓ Adapter Pattern:
  - Unified interface for File/Network/Memory
  - Switch sources without changing pipeline
  - Legacy system integration

✓ Decorator Pattern:
  - Dynamic feature composition
  - Stack processing layers flexibly
  - No subclass explosion
  - Add/remove features at runtime

✓ Combined Power:
  - Flexible data sources (Adapter)
  - Flexible processing (Decorator)
  - Clean, maintainable architecture
```

```
Pipelines: 3
Adapters Used: 3 types
Decorators Used: 5 types
Total Decorator Instances: 8
All operations successful!
```

Tags: `adapter`, `decorator`, `combined-patterns`, `stream`, `pipeline`, `capstone`, `advanced`