CodeForge - B01 - Interface Cơ Bản

Độ khó: ★ Easy

📝 Đề bài

Tạo interface đầu tiên:

- **Interface** `Drawable` với:
  - Abstract method `void draw();` (implicit public abstract)
- Class `Circle` **implements** Drawable với:
  - `double radius`
  - Constructor nhận radius
  - **Implement** draw() in "Drawing circle with radius [r]"

Trong main():

1. **KHÔNG THỂ** tạo `new Drawable()` (interface)
2. Tạo Circle object
3. Gọi draw()
4. Polymorphic reference: `Drawable d = new Circle(5);`

◇ Input

- Một số thực radius

◇ Output

- "Drawing circle with radius [r]"

◇ Constraints

- `0 < radius ≤ 100`

📊 Ví dụ

Test case 1

Input:

```
5.0
```

Output:

```
Drawing circle with radius 5.00
```

Test case 2

Input:

```
3.5
```

Output:

```
Drawing circle with radius 3.50
```

Tags: `interface`, `basic`, `implements`, `cannot-instantiate`

CodeForge - B02 - Implements Keyword

Độ khó: ★ Easy

📝 Đề bài

Class implements interface:

- Interface `Flyable` với:
  - `void fly();`
- Classes `Bird`, `Airplane`, `Superman` implements Flyable với:
  - Different implementations của fly()

Lưu ý: Interface method mặc định là `public abstract`

◇ Input

- Một dòng: Type ("B", "A", hoặc "S")

◇ Output

- Flying message tương ứng

◇ Constraints

- Input chỉ là B, A, hoặc S

📊 Ví dụ

Test case 1

Input:

```
B
```

Output:

```
Bird is flying with wings
```

Test case 2

Input:

```
A
```

Output:

```
Airplane is flying with engines
```

Test case 3

Input:

```
S
```

Output:

```
Superman is flying with superpowers
```

Tags: `interface`, `implements`, `polymorphism`

CodeForge - B03 - Multiple Methods Trong Interface

Độ khó: ★ Easy

📝 Đề bài

Interface với nhiều methods:

- Interface `Movable` với:
  - `void moveUp();`
  - `void moveDown();`
  - `void moveLeft();`
  - `void moveRight();`
- Class `Player` implements Movable với:
  - `int x, y` (position)
  - Implement tất cả 4 methods

◇ Input

- Dòng 1: Initial x, y
- Dòng 2: N (số moves)
- N dòng: Direction (U/D/L/R)

◇ Output

- Final position

◇ Constraints

- `1 ≤ N ≤ 100`
- `-1000 ≤ x, y ≤ 1000`

📊 Ví dụ

Test case 1

Input:

```
0 0
4
U
R
D
L
```

Output:

```
0 0
```

Test case 2

Input:

```
5 5
3
U
U
R
```

Output:

```
6 7
```

Tags: `interface`, `multiple-methods`, `implementation`

CodeForge - B04 - Interface Methods Implicit Public Abstract

Độ khó: ★ Easy

Đề bài

Interface methods mặc định public abstract:

- Interface `Printable` với:
  - `void print();` (không cần public abstract - implicit)
- Class `Document` implements Printable với:
  - **PHẢI** implement với `public void print()` (cannot reduce visibility)

Demo compile error nếu dùng private/protected.

◇ Input

- Một dòng: Document content

◇ Output

- "Printing: [content]"

◇ Constraints

- Độ dài ≤ 200

Ví dụ

Test case 1

Input:

```
Hello World
```

Output:

```
Printing: Hello World
```

Tags: `interface`, `public-abstract`, `implicit`, `visibility`

CodeForge - B05 - Polymorphism Với Interface

Độ khó: ★★ Medium

📝 Đề bài

Interface làm polymorphic reference:

- Interface `Shape` với:
  - `double getArea();`
  - `double getPerimeter();`
- Classes `Circle`, `Rectangle`, `Triangle` implements Shape

Trong main():

1. Tạo `Shape[] shapes` với mixed types
2. Calculate total area polymorphically

◇ Input

- Dòng 1: N
- N dòng: Shape data

◇ Output

- Total area (2 chữ số)

◇ Constraints

- `1 ≤ N ≤ 20`

📊 Ví dụ

Test case 1

Input:

```
3
C 5.0
R 4.0 6.0
T 3.0 4.0 5.0
```

Output:

```
109.54
```

Tags: `interface`, `polymorphism`, `array`, `shapes`

CodeForge - B06 - Interface Không Có Fields (Chỉ Constants)

Độ khó: ★ Easy

📝 Đề bài

Interface chỉ có constants (public static final):

- Interface `MathConstants` với:
  - `double PI = 3.14159;` (implicit public static final)
  - `int MAX_VALUE = 1000;`
- Class `Calculator` implements MathConstants

Lưu ý: Không thể thay đổi giá trị constants

◇ Input

- Một số thực radius

◇ Output

- Area using PI constant

◇ Constraints

- `0 < radius ≤ 100`

📊 Ví dụ

Test case 1

Input:

```
5.0
```

Output:

```
78.54
```

Tags: `interface`, `constants`, `public-static-final`, `implicit`

CodeForge - B07 - Multiple Interface Implementation

Độ khó: ★★ Medium

📝 Đề bài

Class có thể implements nhiều interfaces:

- Interface `Swimmable` với `void swim();`
- Interface `Flyable` với `void fly();`
- Interface `Runnable` với `void run();`
- Class `Duck` implements Swimmable, Flyable với:
  - Implement cả 2 methods
- Class `Penguin` implements Swimmable, Runnable với:
  - Implement cả 2 methods

Multiple inheritance via interfaces!

◇ Input

- Một dòng: Animal type ("D" hoặc "P")

◇ Output

- Abilities của animal

◇ Constraints

- N/A

📊 Ví dụ

Test case 1

Input:

```
D
```

Output:

```
Duck swimming
Duck flying
```

Test case 2

Input:

```
P
```

Output:

```
Penguin swimming
Penguin running
```

Tags: `interface`, `multiple-implementation`, `multiple-inheritance`

CodeForge - B08 - Interface Extending Interface

Độ khó: ★ ★ Medium

📝 Đề bài

Interface có thể extend interface khác:

- Interface `Vehicle` với `void start();`
- Interface `ElectricVehicle` extends Vehicle với:
  - Inherit start()
  - Add `void charge();`
- Class `Tesla` implements ElectricVehicle với:
  - **PHẢI** implement cả 2 methods (start + charge)

◇ Input

- Không có input

◇ Output

- 2 dòng: start, charge

◇ Constraints

- N/A

📊 Ví dụ

Test case 1

Output:

```
Tesla starting
Tesla charging
```

Tags: `interface`, `extending`, `inheritance`, `chain`

CodeForge - B09 - Interface Extending Multiple Interfaces

Độ khó: ★★ Medium

📝 Đề bài

Interface có thể extend NHIỀU interfaces:

- Interface `Readable` với `void read();`
- Interface `Writable` với `void write();`
- Interface `ReadWrite` extends Readable, Writable với:
    - Inherit cả 2 methods
    - Add `void close();`
- Class `File` implements ReadWrite với:
    - Implement tất cả 3 methods

◇ Input

- Không có input

◇ Output

- 3 dòng: read, write, close

◇ Constraints

- N/A

📊 Ví dụ

Test case 1

Output:

```
Reading file
Writing file
Closing file
```

Tags: `interface`, `multiple-extending`, `diamond`, `inheritance`

CodeForge - B10 - Constants Trong Interface

Độ khó: ★★ Medium

📝 Đề bài

Interface constants (public static final implicit):

- Interface `GameConstants` với:
  - `int MAX_PLAYERS = 4;`
  - `int BOARD_SIZE = 10;`
  - `String GAME_NAME = "MyGame";`
- Class `Game` implements GameConstants
- Access constants qua interface name: `GameConstants.MAX_PLAYERS`

◇ Input

- Không có input

◇ Output

- All constants

◇ Constraints

- N/A

📊 Ví dụ

Test case 1

Output:

```
Max Players: 4
Board Size: 10
Game Name: MyGame
```

Tags: `interface`, `constants`, `static-final`, `access`

CodeForge - B11 - Marker Interface

Độ khó: ★★ Medium

📝 Đề bài

Marker interface (empty interface):

- Interface `Serializable` (empty - no methods)
- Classes implement để "mark" special capability
- Class `User` implements Serializable
- Class `Product` implements Serializable

Check `instanceof Serializable` để verify capability.

◇ Input

- Dòng 1: N objects
- N dòng: Type ("U" hoặc "P")

◇ Output

- Count serializable objects

◇ Constraints

- `1 ≤ N ≤ 100`

📊 Ví dụ

Test case 1

Input:

```
4
U
P
U
P
```

Output:

```
4
```

Tags: `interface`, `marker`, `empty`, `capability`

CodeForge - B12 - Interface Constants Best Practices

Độ khó: ★★ Medium

Đề bài

Use interface cho constants (anti-pattern warning):

- Interface `Colors` với:
  - `String RED = "#FF0000";`
  - `String GREEN = "#00FF00";`
  - `String BLUE = "#0000FF";`

Lưu ý: Đây là anti-pattern! Better: enum (Module sau)

Nhưng vẫn widely used trong legacy code.

◇ Input

- Một dòng: Color name (RED/GREEN/BLUE)

◇ Output

- Hex code

◇ Constraints

- N/A

Ví dụ

Test case 1

Input:

```
RED
```

Output:

```
#FF0000
```

Tags: `interface`, `constants`, `anti-pattern`, `legacy`

CodeForge - B13 - Default Methods (Java 8)

Độ khó: ★★ Medium

📝 Đề bài

Interface có thể có default methods (Java 8+):

- Interface `Logger` với:
  - Abstract `void log(String message);`
  - **Default** `default void logError(String message)` có body:
    - log("[ERROR] " + message)
- Class `ConsoleLogger` implements Logger với:
  - Implement log() only
  - Inherit logError() (no need to implement)

◇ Input

- Dòng 1: Normal message
- Dòng 2: Error message

◇ Output

- 2 log messages

◇ Constraints

- Độ dài ≤ 200

📊 Ví dụ

Test case 1

Input:

```
Application started
Connection failed
```

Output:

```
Application started
[ERROR] Connection failed
```

Tags: `interface`, `default-method`, `java8`, `backward-compatible`

CodeForge - B14 - Overriding Default Methods

Độ khó: ★★ Medium

📝 Đề bài

Class có thể override default methods:

- Interface `Formatter` với:
  - **Default** `default String format(String s)` return s.toUpperCase()
- Class `CustomFormatter` implements Formatter với:
  - **Override** format() return s.toLowerCase()
- Class `BasicFormatter` implements Formatter với:
  - Use inherited default (no override)

◇ Input

- Dòng 1: Formatter type ("C" hoặc "B")
- Dòng 2: String

◇ Output

- Formatted string

◇ Constraints

- Độ dài ≤ 100

📊 Ví dụ

Test case 1

Input:

```
C
Hello World
```

Output:

```
hello world
```

Test case 2

Input:

```
B
Hello World
```

Output:

```
HELLO WORLD
```

Tags: `interface`, `default`, `override`, `optional`

CodeForge - B15 - Static Methods Trong Interface (Java 8)

Độ khó: ★★ Medium

📝 Đề bài

Interface có thể có static methods (Java 8+):

- Interface `MathUtils` với:
  - **Static** `static int add(int a, int b)` return a + b
  - **Static** `static int multiply(int a, int b)` return a * b
- Call qua interface name: `MathUtils.add(5, 3)`
- **KHÔNG** inherit vào implementing class

◇ Input

- Dòng 1: a, b

◇ Output

- Dòng 1: Sum
- Dòng 2: Product

◇ Constraints

- `-1000 ≤ a, b ≤ 1000`

📊 Ví dụ

Test case 1

Input:

```
5 3
```

Output:

```
8
15
```

Tags: `interface`, `static-method`, `java8`, `utility`

CodeForge - B16 - Private Methods Trong Interface (Java 9)

Độ khó: ★★★ Hard

📝 Đề bài

Interface có thể có private methods (Java 9+):

- Interface `Validator` với:
  - **Private** `private boolean isNotEmpty(String s)` return s != null && !s.isEmpty()
  - **Default** `default boolean validateName(String name)` use isNotEmpty()
  - **Default** `default boolean validateEmail(String email)` use isNotEmpty() + check @

Private methods = helper methods cho default methods.

◇ Input

- Dòng 1: Name
- Dòng 2: Email

◇ Output

- Validation results

◇ Constraints

- Độ dài ≤ 100

📊 Ví dụ

Test case 1

Input:

```
Alice
alice@example.com
```

Output:

```
Name valid: true
Email valid: true
```

Test case 2

Input:

```


invalid-email
```

Output:

```
Name valid: false
Email valid: false
```

Tags: `interface`, `private-method`, `java9`, `helper`

CodeForge - B17 - Abstract Class Vs Interface - Comparison

Độ khó: ★★★ Hard

📝 Đề bài

So sánh Abstract Class vs Interface:

Abstract Class:

- Can have constructor ✓
- Can have instance fields ✓
- Can have concrete methods ✓
- Single inheritance only ✗

Interface:

- Cannot have constructor ✗
- Only constants (public static final) ✓
- Can have default/static methods (Java 8+) ✓
- Multiple implementation ✓

Tạo 2 examples minh họa differences.

◇ Input

- Type ("ABSTRACT" hoặc "INTERFACE")

◇ Output

- Demo characteristics

◇ Constraints

- N/A

📊 Ví dụ

Test case 1

Input:

```
ABSTRACT
```

Output:

```
Abstract: Has constructor
Abstract: Has fields
Abstract: Single parent only
```

Test case 2

Input:

```
INTERFACE
```

Output:

```
Interface: No constructor
Interface: Constants only
Interface: Multiple implementation
```

Tags: `interface`, `abstract`, `comparison`, `differences`

CodeForge - B18 - When To Use Interface

Độ khó: ★★★ Hard

📝 Đề bài

Khi nào dùng Interface:

- ☑ Định nghĩa contract (behavior)
- ☑ Multiple inheritance cần thiết
- ☑ Không có shared implementation
- ☑ Unrelated classes với common behavior

Example:

- Interface `Comparable` cho sorting
- Interface `Cloneable` cho copying
- Different classes implement same behavior

◇ Input

- Dòng 1: N objects
- N dòng: Type và data

◇ Output

- Sorted objects (using Comparable interface)

◇ Constraints

- `1 ≤ N ≤ 50`

📊 Ví dụ

Test case 1

Input:

```
3
STUDENT Alice 85
STUDENT Bob 92
STUDENT Charlie 78
```

Output:

```
Charlie 78
Alice 85
Bob 92
```

Tags: `interface`, `when-to-use`, `design`, `decision`

CodeForge - B19 - When To Use Abstract Class

Độ khó: ★★★ Hard

📝 Đề bài

Khi nào dùng Abstract Class:

- ✅ Có shared code (concrete methods)
- ✅ Có common fields
- ✅ Related classes (IS-A relationship)
- ✅ Template Method pattern

Example:

- Abstract class `Animal` với eat(), sleep() concrete
- Abstract method makeSound() varies

So sánh với Interface (no shared code).

◇ Input

- Design problem description

◇ Output

- Recommendation: Abstract class or Interface

◇ Constraints

- N/A

📊 Ví dụ

Test case 1

Input:

```
Need shared fields and methods
Related classes
```

Output:

```
Use Abstract Class
```

Test case 2

Input:

```
Unrelated classes
Just define contract
```

Output:

```
Use Interface
```

Tags: `abstract`, `when-to-use`, `design`, `comparison`

CodeForge - B20A - Plugin System Với Interfaces

Độ khó: ★★★ Hard (Advanced)

📝 Đề bài

Tạo plugin architecture:

- Interface `Plugin` với:
  - `String getName();`
  - `String getVersion();`
  - `void initialize();`
  - `void execute();`
  - `void shutdown();`
- Classes `SecurityPlugin`, `LoggingPlugin`, `CachePlugin` implements Plugin
- Class `PluginManager` với:
  - `ArrayList<Plugin> plugins`
  - `void loadPlugin(Plugin p)`
  - `void initializeAll()`
  - `void executeAll()`
  - `void shutdownAll()`

Trong main():

1. Create plugin manager
2. Load N plugins
3. Initialize all
4. Execute all
5. Shutdown all

◇ Input

- Dòng 1: N (plugins)
- N dòng: Plugin type

◇ Output

- Lifecycle log cho tất cả plugins

◇ Constraints

- `1 ≤ N ≤ 10`

📊 Ví dụ

Test case 1

Input:

```
3
SECURITY
LOGGING
CACHE
```

Output:

```
Loading: SecurityPlugin v1.0
Loading: LoggingPlugin v1.0
Loading: CachePlugin v1.0

Initializing all plugins...
SecurityPlugin initialized
LoggingPlugin initialized
CachePlugin initialized

Executing all plugins...
SecurityPlugin running
LoggingPlugin running
CachePlugin running

Shutting down all plugins...
SecurityPlugin stopped
LoggingPlugin stopped
CachePlugin stopped
```

Tags: `interface`, `plugin`, `architecture`, `lifecycle`, `advanced`

CodeForge - B21A - Event Listener System

Độ khó: ★★★ Hard (Advanced)

📝 Đề bài

Tạo event system với interfaces:

- Interface `EventListener` với:
  - `void onEvent(String eventType, String data);`
- Interface `EventSource` với:
  - `void addEventListener(EventListener listener);`
  - `void removeEventListener(EventListener listener);`
  - `void fireEvent(String eventType, String data);`
- Class `Button` implements EventSource
- Classes `ClickLogger`, `ClickCounter`, `ClickNotifier` implements EventListener

Observer pattern với interfaces!

◇ Input

- Dòng 1: N (listeners)
- N dòng: Listener types
- Dòng N+2: M (clicks)
- M dòng: Click data

◇ Output

- Event notifications tới tất cả listeners

◇ Constraints

- `1 ≤ N ≤ 10`
- `1 ≤ M ≤ 50`

📊 Ví dụ

Test case 1

Input:

```
3
LOGGER
COUNTER
NOTIFIER
3
x=10 y=20
```

```
x=30 y=40
x=50 y=60
```

Output:

```

Logger: Click at x=10 y=20
Counter: Total clicks = 1
Notifier: Alert! Button clicked

Logger: Click at x=30 y=40
Counter: Total clicks = 2
Notifier: Alert! Button clicked

Logger: Click at x=50 y=60
Counter: Total clicks = 3
Notifier: Alert! Button clicked
```

Tags: `interface`, `event`, `listener`, `observer`, `pattern`, `advanced`

CodeForge - B22A - Strategy Pattern Với Interfaces

Độ khó: ★★★ Hard (Advanced)

📝 Đề bài

Tạo strategy pattern:

- Interface `PaymentStrategy` với:
  - `boolean pay(double amount);`
  - `String getPaymentType();`
- Classes `CreditCardStrategy`, `PayPalStrategy`, `CryptoStrategy` implements PaymentStrategy
- Class `ShoppingCart` với:
  - `ArrayList<Item> items`
  - `PaymentStrategy paymentStrategy`
  - `void setPaymentStrategy(PaymentStrategy strategy)`
  - `void checkout()`

Trong main():

1. Add items to cart
2. Try different payment strategies
3. Process checkout

◇ Input

- Dòng 1: N (items)
- N dòng: Item name, price
- Dòng N+2: Payment type

◇ Output

- Cart summary
- Payment processing

◇ Constraints

- `1 ≤ N ≤ 20`

📊 Ví dụ

Test case 1

Input:

```
3
Laptop 1000.00
```

```
Mouse 25.00
Keyboard 75.00
CREDITCARD
```

Output:

```

Shopping Cart:
- Laptop: $1000.00
- Mouse: $25.00
- Keyboard: $75.00
Total: $1100.00

Processing with Credit Card...
Payment successful!
```

Tags: `interface`, `strategy`, `pattern`, `payment`, `advanced`

CodeForge - B23A - Sorting Algorithm Interface

Độ khó: ★★★ Hard (Advanced)

📝 Đề bài

Tạo sorting framework với interfaces:

- Interface `Sorter` với:
  - `void sort(int[] arr);`
  - `String getAlgorithmName();`
  - `default void printArray(int[] arr)` print array
- Classes `BubbleSort`, `QuickSort`, `MergeSort`, `InsertionSort` implements Sorter
- Class `SortingBenchmark` với:
  - `void benchmark(Sorter sorter, int[] data)`
  - Measure performance

◇ Input

- Dòng 1: N
- Dòng 2: N numbers
- Dòng 3: M (algorithms to test)
- M dòng: Algorithm names

◇ Output

- Sorted arrays
- Performance comparison

◇ Constraints

- `1 ≤ N ≤ 100`

📊 Ví dụ

Test case 1

Input:

```
10
9 5 2 8 1 7 3 6 4 0
3
BUBBLE
QUICK
MERGE
```

Output:

```
Bubble Sort:
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Time: 0.05ms

Quick Sort:
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Time: 0.02ms

Merge Sort:
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Time: 0.03ms

Fastest: Quick Sort
```

Tags: `interface`, `sorting`, `algorithm`, `benchmark`, `advanced`

CodeForge - B24A - Data Persistence Interface

Độ khó: ★★★ Hard (Advanced)

📝 Đề bài

Tạo persistence layer với interfaces:

- Interface `Repository<T>` với:
  - `void save(T entity);`
  - `T findById(String id);`
  - `List<T> findAll();`
  - `void delete(String id);`
- Classes `FileRepository`, `DatabaseRepository`, `MemoryRepository` implements Repository
  - Different storage mechanisms
- Class `User` với id, name, email

Trong main():

1. Create repositories
2. Save entities
3. Find và delete
4. Compare different implementations

◇ Input

- Dòng 1: Repository type
- Dòng 2: N (operations)
- N dòng: Operation (SAVE/FIND/DELETE) và data

◇ Output

- Operation results

◇ Constraints

- `1 ≤ N ≤ 50`

📊 Ví dụ

Test case 1

Input:

```
MEMORY
5
SAVE U001 Alice alice@email.com
SAVE U002 Bob bob@email.com
FIND U001
DELETE U002
FINDALL
```

Output:

```
[Memory] Saved: U001 - Alice
[Memory] Saved: U002 - Bob
[Memory] Found: U001 - Alice (alice@email.com)
[Memory] Deleted: U002
[Memory] All users:
- U001: Alice (alice@email.com)
```

Tags: `interface`, `repository`, `persistence`, `crud`, `advanced`

CodeForge - B25A - Complete Interface System - Notification Framework

Độ khó: ★★★ Hard (Advanced)

📝 Đề bài

Tạo complete notification framework:

- Interface `NotificationChannel` với:
  - `boolean send(String recipient, String message);`
  - `String getChannelName();`
  - `default boolean isAvailable()` return true (hook)
- Interface `NotificationFormatter` với:
  - `String format(String message, String priority);`
- Classes `EmailChannel`, `SMSChannel`, `PushChannel`, `SlackChannel` implements NotificationChannel
- Classes `PlainFormatter`, `HTMLFormatter`, `MarkdownFormatter` implements NotificationFormatter
- Class `Notification` với:
  - `String message, priority, recipient`
- Class `NotificationService` với:
  - `List<NotificationChannel> channels`
  - `NotificationFormatter formatter`
  - Strategy pattern: set formatter dynamically
  - `void broadcast(Notification notification)` send via all channels
  - `void sendVia(NotificationChannel channel, Notification notification)`

Trong main():

1. Configure notification service
2. Add multiple channels
3. Set formatter
4. Send notifications
5. Handle failures gracefully
6. Generate delivery report

◇ Input

- Dòng 1: N (channels)
- N dòng: Channel types
- Dòng N+2: Formatter type
- Dòng N+3: M (notifications)
- M dòng: Priority, recipient, message

◇ Output

- Delivery log

- Summary report

◇ Constraints

- `1 ≤ N ≤ 5`
- `1 ≤ M ≤ 20`

📊 Ví dụ

Test case 1

Input:

```
4
EMAIL
SMS
PUSH
SLACK
HTML
3
HIGH admin@company.com Server down
MEDIUM user@company.com Update available
LOW team@company.com Daily report
```

Output:

```
=== Notification Service Started ===
Channels: Email, SMS, Push, Slack
Formatter: HTML

Broadcasting: HIGH priority to admin@company.com
[Email] ✓ Sent: <b>HIGH</b> Server down
[SMS] ✓ Sent: <b>HIGH</b> Server down
[Push] ✓ Sent: <b>HIGH</b> Server down
[Slack] ✓ Sent: <b>HIGH</b> Server down

Broadcasting: MEDIUM priority to user@company.com
[Email] ✓ Sent: <b>MEDIUM</b> Update available
[SMS] ✓ Sent: <b>MEDIUM</b> Update available
[Push] ✓ Sent: <b>MEDIUM</b> Update available
[Slack] ✓ Sent: <b>MEDIUM</b> Update available

Broadcasting: LOW priority to team@company.com
[Email] ✓ Sent: <b>LOW</b> Daily report
[SMS] ✓ Sent: <b>LOW</b> Daily report
[Push] ✓ Sent: <b>LOW</b> Daily report
[Slack] ✓ Sent: <b>LOW</b> Daily report

=== Delivery Report ===
Total Notifications: 3
Total Deliveries: 12
Success Rate: 100%
Channels Used: 4
```

Tags: `interface`, `notification`, `complete-system`, `strategy`, `observer`, `capstone`, `advanced`