CodeForge - B01 - Telescoping Constructor Problem

Độ khó: ★ Easy

📝 Đề bài

Demo vấn đề Telescoping Constructor:

- Class `Pizza` với nhiều optional parameters:
  - `String size` (required)
  - `boolean cheese` (optional)
  - `boolean pepperoni` (optional)
  - `boolean bacon` (optional)
  - `boolean olives` (optional)

Problem: Cần nhiều constructors!

```
Pizza(String size)
Pizza(String size, boolean cheese)
Pizza(String size, boolean cheese, boolean pepperoni)
Pizza(String size, boolean cheese, boolean pepperoni, boolean bacon)
// ... rất nhiều combinations!
```

Tạo 4 constructors và demo confusion.

◇ Input

- Dòng 1: Size
- Dòng 2-5: Toppings (YES/NO)

◇ Output

- Pizza description

◇ Constraints

- Size: SMALL/MEDIUM/LARGE

📊 Ví dụ

Test case 1

Input:

```
LARGE
YES
YES
```

```
NO
YES
```

Output:

```

Pizza: LARGE
Cheese: YES
Pepperoni: YES
Bacon: NO
Olives: YES
[Problem: Nhiều constructors, dễ nhầm lẫn parameter order!]
```

Tags: `telescoping-constructor`, `problem`, `anti-pattern`

CodeForge - B02 - Builder Pattern Cơ Bản

Độ khó: ★★ Medium

📝 Đề bài

Giải quyết Telescoping Constructor bằng Builder:

- Class `Computer` với:
    - Required: `String CPU`, `String RAM`
    - Optional: `String storage`, `String GPU`, `boolean wifi`
- **Static nested class** `Builder` với:
    - Required params trong constructor
    - Optional params qua setter methods
    - `Computer build()` return new Computer(this)
- Private constructor `Computer(Builder builder)`

Builder pattern: Separate construction from representation

◇ Input

- Dòng 1-2: CPU, RAM (required)
- Dòng 3-5: Storage, GPU, WiFi (optional, "NONE" = skip)

◇ Output

- Computer specs

◇ Constraints

- N/A

📊 Ví dụ

Test case 1

Input:

```
Intel_i7
16GB
512GB_SSD
RTX_3060
YES
```

Output:

```
Computer Specs:
CPU: Intel_i7
RAM: 16GB
Storage: 512GB_SSD
GPU: RTX_3060
WiFi: YES
```

Test case 2

Input:

```
AMD_Ryzen5
8GB
NONE
NONE
NO
```

Output:

```
Computer Specs:
CPU: AMD_Ryzen5
RAM: 8GB
Storage: Not specified
GPU: Not specified
WiFi: NO
```

Tags: `builder`, `pattern`, `nested-class`, `solution`

CodeForge - B03 - Method Chaining Trong Builder

Độ khó: ★ ★ Medium

📝 Đề bài

Builder với method chaining (fluent interface):

- Class `Car` với fields: brand, model, color, sunroof, GPS
- Builder class với methods return `this`:

  ```
  public Builder setBrand(String brand) {
      this.brand = brand;
      return this;
  }
  ```

Cho phép:

```
Car car = new Car.Builder()
    .setBrand("Toyota")
    .setModel("Camry")
    .setColor("Blue")
    .build();
```

◇ Input

- Dòng 1: Brand
- Dòng 2: Model
- Dòng 3: Color
- Dòng 4: Sunroof (YES/NO)
- Dòng 5: GPS (YES/NO)

◇ Output

- Car details

◇ Constraints

- N/A

📊 Ví dụ

Test case 1

Input:

```
Toyota
Camry
Blue
YES
YES
```

Output:

```
Car: Toyota Camry
Color: Blue
Sunroof: YES
GPS: YES
[Built using fluent interface!]
```

Tags: `builder`, `method-chaining`, `fluent-interface`, `return-this`

CodeForge - B04 - Fluent API Builder

Độ khó: ★★ Medium

📝 Đề bài

Tạo expressive fluent API:

- Class `EmailMessage` với:
  - `String from, to, subject, body`
  - `List<String> attachments`
- Builder với readable methods:
  - `from(String email)`
  - `to(String email)`
  - `subject(String text)`
  - `body(String content)`
  - `attach(String file)`
  - `build()`

Usage:

```
EmailMessage email = new EmailMessage.Builder()
    .from("sender@email.com")
    .to("receiver@email.com")
    .subject("Hello")
    .body("Hi there!")
    .attach("file.pdf")
    .build();
```

◇ Input

- Dòng 1: From
- Dòng 2: To
- Dòng 3: Subject
- Dòng 4: Body
- Dòng 5: N (attachments)
- N dòng: Filenames

◇ Output

- Email summary

◇ Constraints

- `0 ≤ N ≤ 10`

📊 Ví dụ

Test case 1

Input:

```
alice@email.com
bob@email.com
Project Update
Please review the attached files
2
report.pdf
data.xlsx
```

Output:

```
Email Message:
From: alice@email.com
To: bob@email.com
Subject: Project Update
Body: Please review the attached files
Attachments: report.pdf, data.xlsx
```

Tags: `builder`, `fluent-api`, `readable`, `expressive`

CodeForge - B05 - Builder Với Validation

Độ khó: ★★★ Hard

📝 Đề bài

Builder với validation logic:

- Class `User` với:
  - Required: `String username`, `String password`
  - Optional: `String email`, `int age`
- Builder với validation trong `build()`:
  - Username: 3-20 chars
  - Password: min 8 chars
  - Email: contain @
  - Age: 13-120
  - Throw `IllegalStateException` nếu invalid

◇ Input

- Dòng 1: Username
- Dòng 2: Password
- Dòng 3: Email
- Dòng 4: Age

◇ Output

- User created hoặc validation error

◇ Constraints

- N/A

📊 Ví dụ

Test case 1

Input:

```
alice
password123
alice@email.com
25
```

Output:

```
User created successfully:
Username: alice
Email: alice@email.com
Age: 25
```

Test case 2

Input:

```
ab
pass
invalid-email
10
```

Output:

```
Validation errors:
- Username too short (min 3 chars)
- Password too short (min 8 chars)
- Invalid email format
- Age below minimum (13)
```

Tags: `builder`, `validation`, `error-handling`, `constraints`

CodeForge - B06 - When To Use Builder Pattern

Độ khó: ★★ Medium

📝 Đề bài

Khi nào dùng Builder:

- ✅ Nhiều parameters (>5)
- ✅ Nhiều optional parameters
- ✅ Immutable objects
- ✅ Need readable construction
- ❌ Few parameters → normal constructor OK
- ❌ Mutable objects → setters OK

Demo 2 scenarios: Builder needed vs Normal constructor OK.

🔹 Input

- Scenario type (BUILDER/NORMAL)

🔹 Output

- Recommendation

🔹 Constraints

- N/A

📊 Ví dụ

Test case 1

Input:

```
BUILDER
```

Output:

```
Scenario: Complex object with 10+ parameters
Recommendation: Use Builder Pattern
Reasons:
- Many optional parameters
- Improved readability
- Immutable object
- Validation logic
```

Test case 2

Input:

```
NORMAL
```

Output:

```
Scenario: Simple object with 2-3 parameters
Recommendation: Use Normal Constructor
Reasons:
- Few parameters
- All required
- Simple validation
- Builder overhead unnecessary
```

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

CodeForge - B07 - Prototype Pattern - Cloneable Interface

Độ khó: ★★ Medium

📝 Đề bài

Tạo cloneable object:

- Class `Shape` implements `Cloneable` với:
  - `String type`
  - `String color`
  - `@Override public Object clone()`:
    - try { return super.clone(); }
    - catch (CloneNotSupportedException) { return null; }

Prototype pattern: Create new object by copying existing

◇ Input

- Dòng 1: Original type, color
- Dòng 2: N (clones)

◇ Output

- Original + N clones

◇ Constraints

- `1 ≤ N ≤ 10`

📊 Ví dụ

Test case 1

Input:

```
Circle Red
3
```

Output:

```
Original: Circle (Red) @hash1
Clone 1: Circle (Red) @hash2
Clone 2: Circle (Red) @hash3
Clone 3: Circle (Red) @hash4
[All clones are separate objects]
```

Tags: `prototype`, `cloneable`, `interface`, `clone`, `pattern`

CodeForge - B08 - Shallow Copy Demo

Độ khó: ★★ Medium

📝 Đề bài

Demo shallow copy problem:

- Class `Person` implements Cloneable với:
  - `String name` (primitive/immutable)
  - `Address address` (mutable object)
- Class `Address` với:
  - `String city`

Shallow copy: Clone person nhưng address vẫn reference cùng object!

Modify clone's address → affects original!

◇ Input

- Dòng 1: Name
- Dòng 2: City

◇ Output

- Demo shallow copy problem

◇ Constraints

- N/A

📊 Ví dụ

Test case 1

Input:

```
Alice
New_York
```

Output:

```
Original: Alice in New_York
Clone: Alice in New_York

[Changing clone's city to London...]
```

```
PROBLEM - Shallow Copy:
Original: Alice in London (CHANGED!)
Clone: Alice in London
[Both share same Address object!]
```

Tags: `prototype`, `shallow-copy`, `problem`, `reference`

CodeForge - B09 - Deep Copy Solution

Độ khó: ★★★ Hard

📝 Đề bài

Giải quyết shallow copy bằng deep copy:

- Class `Employee` implements Cloneable với:
  - `String name`
  - `Department department` (mutable object)
- Override clone() để deep copy:

  ```java
  @Override
  public Object clone() {
      Employee cloned = (Employee) super.clone();
      cloned.department = (Department) department.clone();
      return cloned;
  }
  ```

Deep copy: Clone tất cả nested objects!

◇ Input

- Dòng 1: Name
- Dòng 2: Department name

◇ Output

- Demo deep copy working correctly

◇ Constraints

- N/A

📊 Ví dụ

Test case 1

Input:

```
Bob
Engineering
```

Output:

```
Original: Bob in Engineering
Clone: Bob in Engineering

[Changing clone's department to Sales...]

SOLUTION - Deep Copy:
Original: Bob in Engineering (UNCHANGED!)
Clone: Bob in Sales
[Separate Department objects!]
```

Tags: `prototype`, `deep-copy`, `solution`, `nested-clone`

CodeForge - B10 - Prototype Registry

Độ khó: ★★★ Hard

📝 Đề bài

Tạo Prototype Registry pattern:

- Class `ShapeCache` (Singleton) với:
  - `Map<String, Shape> shapeMap`
  - `void loadCache()` pre-load prototypes
  - `Shape getShape(String type)` return clone

Registry: Store và clone prototypes

◇ Input

- Dòng 1: N (shapes to create)
- N dòng: Shape type (CIRCLE/RECTANGLE/TRIANGLE)

◇ Output

- Cloned shapes

◇ Constraints

- `1 ≤ N ≤ 20`

📊 Ví dụ

Test case 1

Input:

```
4
CIRCLE
RECTANGLE
CIRCLE
TRIANGLE
```

Output:

```
[Registry] Loading prototypes...
  ✓ Circle prototype loaded
  ✓ Rectangle prototype loaded
  ✓ Triangle prototype loaded
```

```
Creating shapes from registry:
Shape 1: Circle (cloned from prototype)
Shape 2: Rectangle (cloned from prototype)
Shape 3: Circle (cloned from prototype)
Shape 4: Triangle (cloned from prototype)

Total clones created: 4
```

Tags: `prototype`, `registry`, `cache`, `pattern`

CodeForge - B11 - When To Use Prototype Pattern

Độ khó: ⭐⭐ Medium

📝 Đề bài

Khi nào dùng Prototype:

- ✅ Object creation expensive (DB query, network call)
- ✅ Many similar objects needed
- ✅ Avoid subclassing
- ✅ Runtime object composition
- ❌ Simple objects → new keyword OK
- ❌ Deep copy complex/error-prone

Demo scenarios: Prototype needed vs Normal creation.

🔹 Input

- Scenario (PROTOTYPE/NORMAL)

🔹 Output

- Recommendation

🔹 Constraints

- N/A

📊 Ví dụ

Test case 1

Input:

```
PROTOTYPE
```

Output:

```
Scenario: Creating 1000 similar game objects
Recommendation: Use Prototype Pattern
Reasons:
- Expensive initialization (load textures, animations)
- Many similar instances needed
- Better performance than new keyword
- Clone faster than reconstruct
```

Test case 2

Input:

```
NORMAL
```

Output:

```
Scenario: Creating simple DTOs
Recommendation: Use Normal Constructor
Reasons:
- Lightweight objects
- No expensive initialization
- Prototype overhead unnecessary
- Clone complexity not worth it
```

Tags: `prototype`, `when-to-use`, `design-decision`, `performance`

CodeForge - B12A - Complete Builder System - Query Builder

Độ khó: ★★★ Hard (Advanced)

📝 Đề bài

Tạo SQL Query Builder:

- Class `SQLQuery` (immutable) với:
  - `String table`
  - `List<String> columns`
  - `List<String> conditions`
  - `String orderBy`
  - `Integer limit`
  - Method `String toSQL()` generate query
- **Static nested Builder** với fluent API:
  - `select(String... columns)`
  - `from(String table)`
  - `where(String condition)`
  - `and(String condition)`
  - `or(String condition)`
  - `orderBy(String column, String direction)`
  - `limit(int n)`
  - `build()` with validation

Trong main():

1. Build N queries với different conditions
2. Generate SQL strings
3. Execute (simulate)

◇ Input

- Dòng 1: N (queries)
- N nhóm query specifications:
  - Table name
  - Columns (comma-separated)
  - M conditions
  - Order by (column, direction)
  - Limit

◇ Output

- Generated SQL queries

◇ Constraints

- `1 ≤ N ≤ 10`

📊 Ví dụ

Test case 1

Input:

```
2
users
id,name,email
2
age > 18
status = 'active'
created_at DESC
10
products
*
1
price < 100
name ASC
20
```

Output:

```
=== Query Builder ===

Query 1:
SELECT id, name, email
FROM users
WHERE age > 18 AND status = 'active'
ORDER BY created_at DESC
LIMIT 10

Query 2:
SELECT *
FROM products
WHERE price < 100
ORDER BY name ASC
LIMIT 20

[Both queries built using Fluent Builder API]
Validation: ✓ All queries valid
```

Tags: `builder`, `sql`, `query`, `fluent-api`, `immutable`, `advanced`

CodeForge - B13A - Complete Builder System - HTTP Request Builder

Độ khó: ★★★ Hard (Advanced)

📝 Đề bài

Tạo HTTP Request Builder:

- Class `HTTPRequest` (immutable) với:
  - `String method` (GET/POST/PUT/DELETE)
  - `String url`
  - `Map<String, String> headers`
  - `Map<String, String> queryParams`
  - `String body`
  - `int timeout`
- Builder với validation:
  - Required: method, url
  - Optional: headers, params, body, timeout
  - Validation:
    - URL must start with http:// or https://
    - Timeout > 0
    - POST/PUT require body
    - GET/DELETE cannot have body
- Method `String execute()` simulate request

Trong main():

1. Build N requests
2. Validate
3. Execute
4. Handle errors

◇ Input

- Dòng 1: N (requests)
- N nhóm request data

◇ Output

- Request execution log

◇ Constraints

- `1 ≤ N ≤ 10`

📊 Ví dụ

Test case 1

Input:

```
3
GET https://api.example.com/users 5000
POST https://api.example.com/users 3000 {"name":"Alice"}
DELETE https://api.example.com/users/123 2000
```

Output:

```
=== HTTP Request Builder ===

Request 1:
Method: GET
URL: https://api.example.com/users
Timeout: 5000ms
Headers: Accept: application/json
[Executing...]
✓ 200 OK - Response received

Request 2:
Method: POST
URL: https://api.example.com/users
Timeout: 3000ms
Body: {"name":"Alice"}
Headers: Content-Type: application/json
[Executing...]
✓ 201 Created - User created

Request 3:
Method: DELETE
URL: https://api.example.com/users/123
Timeout: 2000ms
[Executing...]
✓ 204 No Content - User deleted

=== Summary ===
Total Requests: 3
Successful: 3
Failed: 0
All built using Builder Pattern with validation
```

Tags: `builder`, `http`, `request`, `validation`, `fluent-api`, `advanced`

CodeForge - B14A - Complete Prototype System - Game Entity Cloning

Độ khó: ★★★ Hard (Advanced)

📝 Đề bài

Tạo game entity cloning system:

- Abstract class `GameEntity` implements Cloneable với:
  - `String id`
  - `int x, y` (position)
  - `Sprite sprite` (image data - expensive to load)
  - `Stats stats` (health, damage, speed)
  - `abstract void update();`
  - `@Override public GameEntity clone()` (deep copy)
- Classes `Enemy`, `PowerUp`, `Obstacle` extends GameEntity
- Class `EntityPrototypeRegistry` với:
  - Pre-load entity prototypes với sprites
  - Clone entities (fast - sprite already loaded)
  - Track cloning statistics
- Class `EntityManager` với:
  - Spawn entities from prototypes
  - Position clones
  - Update all entities

Trong main():

1. Load prototypes (expensive - load sprites)
2. Spawn N waves of entities (fast - clone)
3. Update positions
4. Show performance comparison:
   - Time to create from scratch
   - Time to clone from prototype

◇ Input

- Dòng 1: N (waves)
- N dòng: M entities, types

◇ Output

- Spawning log
- Performance metrics

◇ Constraints

- `1 ≤ N ≤ 5`
- `1 ≤ M ≤ 50`

📊 Ví dụ

Test case 1

Input:

```
3
5 ENEMY POWERUP ENEMY OBSTACLE ENEMY
3 POWERUP POWERUP OBSTACLE
4 ENEMY ENEMY POWERUP POWERUP
```

Output:

```
=== Game Entity Prototype System ===

[Loading Prototypes - Expensive Operation]
Loading Enemy sprite... (500ms)
Loading PowerUp sprite... (300ms)
Loading Obstacle sprite... (400ms)
Total loading time: 1200ms

=== Spawning Wave 1 (5 entities) ===
Cloning Enemy #1 at (10, 20) - 2ms
Cloning PowerUp #1 at (30, 40) - 1ms
Cloning Enemy #2 at (50, 60) - 2ms
Cloning Obstacle #1 at (70, 80) - 2ms
Cloning Enemy #3 at (90, 100) - 2ms
Wave 1 spawn time: 9ms

=== Spawning Wave 2 (3 entities) ===
Cloning PowerUp #2 at (15, 25) - 1ms
Cloning PowerUp #3 at (35, 45) - 1ms
Cloning Obstacle #2 at (55, 65) - 2ms
Wave 2 spawn time: 4ms

=== Spawning Wave 3 (4 entities) ===
Cloning Enemy #4 at (20, 30) - 2ms
Cloning Enemy #5 at (40, 50) - 2ms
Cloning PowerUp #4 at (60, 70) - 1ms
Cloning PowerUp #5 at (80, 90) - 1ms
Wave 3 spawn time: 6ms

=== Performance Analysis ===

If created from scratch (load sprite each time):
  Wave 1: ~1500ms (5 entities × 300ms avg)
  Wave 2: ~900ms (3 entities × 300ms avg)
  Wave 3: ~1200ms (4 entities × 300ms avg)
  Total: ~3600ms

Using Prototype Pattern:
  Initial loading: 1200ms (one-time)
  Wave 1: 9ms (clone)
  Wave 2: 4ms (clone)
  Wave 3: 6ms (clone)
  Total: 1219ms

Performance Gain: 66% faster! (2381ms saved)
Clones created: 12
All share pre-loaded sprites (deep copy of stats only)
```

Tags: `prototype`, `game`, `entity`, `performance`, `deep-copy`, `advanced`

CodeForge - B15A - Complete System - Document Builder & Cloning

Độ khó: ★★★ Hard (Advanced)

📝 Đề bài

Tạo complete document system kết hợp Builder + Prototype:

- Class `Document` (immutable) implements Cloneable với:
  - `String title, author`
  - `List<Section> sections`
  - `Map<String, String> metadata`
  - `DocumentStyle style`
  - `Timestamp created`
- **Builder pattern** tạo documents:
  - Fluent API
  - Validation
  - Default values
- **Prototype pattern** clone documents:
  - Deep copy sections
  - Update metadata
  - New timestamp
- Class `DocumentTemplate` với:
  - Pre-defined document templates
  - Clone + customize
- Class `DocumentManager` với:
  - Template registry
  - Version control
  - Export functionality

Trong main():

1. Create templates using Builder
2. Clone templates for new documents
3. Customize clones
4. Track versions
5. Export documents
6. Show comparison: Builder (slow) vs Clone (fast)

◇ Input

- Dòng 1: N (templates)
- Templates data
- Dòng X: M (documents from templates)
- Documents customization data

◇ Output

- Document creation log
- Version tracking
- Performance metrics

◇ Constraints

- `1 ≤ N ≤ 5`
- `1 ≤ M ≤ 20`

📊 Ví dụ

Test case 1

Input:

```
2
Report Standard_Report_Template John_Doe 3
Introduction
Methodology
Conclusion
Proposal Business_Proposal_Template Jane_Smith 2
Executive_Summary
Budget
3
Report Q1_Sales_Report Alice Johnson
Report Q2_Sales_Report Bob Wilson
Proposal Partnership_Proposal Charlie Brown
```

Output:

```
=== Document Management System ===
Combining Builder + Prototype Patterns

[Creating Templates Using Builder Pattern]

Template 1: Standard_Report_Template
  Building... (using Builder pattern)
  ✓ Title: Report
  ✓ Author: John_Doe
  ✓ Sections: 3
    - Introduction
    - Methodology
    - Conclusion
  ✓ Style: Default
  ✓ Metadata: Template=true
  Build time: 45ms
```

```
Template 2: Business_Proposal_Template
  Building... (using Builder pattern)
  ✓ Title: Proposal
  ✓ Author: Jane_Smith
  ✓ Sections: 2
    - Executive_Summary
    - Budget
  ✓ Style: Business
  ✓ Metadata: Template=true
  Build time: 38ms

[Templates Saved to Registry]

=== Creating Documents from Templates (Prototype) ===

Document 1: Q1_Sales_Report
  Cloning from: Standard_Report_Template
  ✓ Deep copy sections (3 sections)
  ✓ Updating author: Alice Johnson
  ✓ New timestamp: 2024-12-22 10:30:00
  ✓ Version: 1.0
  Clone + customize time: 8ms

Document 2: Q2_Sales_Report
  Cloning from: Standard_Report_Template
  ✓ Deep copy sections (3 sections)
  ✓ Updating author: Bob Wilson
  ✓ New timestamp: 2024-12-22 10:30:01
  ✓ Version: 1.0
  Clone + customize time: 7ms

Document 3: Partnership_Proposal
  Cloning from: Business_Proposal_Template
  ✓ Deep copy sections (2 sections)
  ✓ Updating author: Charlie Brown
  ✓ New timestamp: 2024-12-22 10:30:02
  ✓ Version: 1.0
  Clone + customize time: 6ms

[Exporting Documents]
✓ Q1_Sales_Report.pdf exported
✓ Q2_Sales_Report.pdf exported
✓ Partnership_Proposal.pdf exported

=== Performance Comparison ===

Using Builder for all documents:
  Template 1: 45ms
  Template 2: 38ms
  Document 1: ~45ms (rebuild from scratch)
  Document 2: ~45ms (rebuild from scratch)
  Document 3: ~38ms (rebuild from scratch)
  Total: ~211ms
```

```
Using Builder + Prototype:
  Template 1: 45ms (one-time)
  Template 2: 38ms (one-time)
  Document 1: 8ms (clone + customize)
  Document 2: 7ms (clone + customize)
  Document 3: 6ms (clone + customize)
  Total: 104ms

Performance Gain: 51% faster! (107ms saved)

=== Pattern Benefits ===
✓ Builder: Clean construction of complex templates
✓ Prototype: Fast document generation from templates
✓ Immutability: Thread-safe, version-safe
✓ Deep Copy: Independent document instances
✓ Fluent API: Readable, maintainable code

Total Templates: 2
Total Documents: 3
All documents validated and exported successfully
```

Tags: `builder`, `prototype`, `combined-patterns`, `document`, `complete-system`, `capstone`, `advanced`