CodeForge - B01 - Singleton Pattern - Eager Initialization

Độ khó: ★ Easy

📝 Đề bài

Tạo Singleton với Eager Initialization:

- Class `DatabaseConnection` với:
  - `private static final DatabaseConnection instance = new DatabaseConnection();` (eager)
  - **Private constructor** (ngăn external instantiation)
  - `public static DatabaseConnection getInstance()` return instance
  - Method `void connect()` in "Connected to database"

Eager: Instance tạo ngay lúc class load (thread-safe by default)

Trong main():

1. Gọi getInstance() nhiều lần
2. Verify cùng instance

◇ Input

- Dòng 1: N (số lần gọi getInstance)

◇ Output

- Connection message
- "All references point to same instance: true"

◇ Constraints

- `1 ≤ N ≤ 10`

📊 Ví dụ

Test case 1

Input:

```
3
```

Output:

```
Connected to database
Connected to database
Connected to database
All references point to same instance: true
```

Tags: `singleton`, `eager`, `initialization`, `thread-safe`, `pattern`

CodeForge - B02 - Singleton Pattern - Lazy Initialization

Độ khó: ⭐ Easy

📝 Đề bài

Tạo Singleton với Lazy Initialization:

- Class `Logger` với:
  - `private static Logger instance = null;` (not initialized)
  - Private constructor
  - `public static Logger getInstance()`:
    - if (instance == null) instance = new Logger()
    - return instance
  - Method `void log(String message)`

Lazy: Instance tạo khi cần (first call to getInstance)

Lưu ý: NOT thread-safe (single-threaded OK)

◇ Input

- Dòng 1: N (messages)
- N dòng: Log messages

◇ Output

- N log messages

◇ Constraints

- `1 ≤ N ≤ 20`

📊 Ví dụ

Test case 1

Input:

```
3
Application started
Processing data
Application stopped
```

Output:

```
[Logger] Application started
[Logger] Processing data
[Logger] Application stopped
```

Tags: `singleton`, `lazy`, `initialization`, `not-thread-safe`, `pattern`

CodeForge - B03 - Singleton Pattern - Thread-Safe (Synchronized)

Độ khó: ★★ Medium

📝 Đề bài

Tạo thread-safe Singleton:

- Class `ConfigManager` với:
  - `private static ConfigManager instance = null;`
  - Private constructor
  - `public static synchronized ConfigManager getInstance()`:
    - synchronized keyword ensures thread-safety
    - if (instance == null) create
    - return instance

Thread-safe: Synchronized method (simple but slow)

◇ Input

- Dòng 1: N (config operations)
- N dòng: Config key-value pairs

◇ Output

- Config stored messages

◇ Constraints

- `1 ≤ N ≤ 50`

📊 Ví dụ

Test case 1

Input:

```
3
DB_HOST localhost
DB_PORT 3306
DB_NAME mydb
```

Output:

```
[Config] Stored: DB_HOST = localhost
[Config] Stored: DB_PORT = 3306
[Config] Stored: DB_NAME = mydb
```

Tags: `singleton`, `thread-safe`, `synchronized`, `performance`, `pattern`

CodeForge - B04 - Singleton Pattern - Double-Checked Locking

Độ khó: ★★★ Hard

📝 Đề bài

Tạo Singleton với Double-Checked Locking:

- Class `CacheManager` với:
  - `private static volatile CacheManager instance = null;` (volatile important!)
  - Private constructor
  - `public static CacheManager getInstance()`:

    ```
    if (instance == null) {
        synchronized (CacheManager.class) {
            if (instance == null) {
                instance = new CacheManager();
            }
        }
    }
    return instance;
    ```

Double-check: Optimize synchronized (only when needed)

◇ Input

- Dòng 1: N (cache operations)
- N dòng: PUT/GET key [value]

◇ Output

- Cache operation results

◇ Constraints

- `1 ≤ N ≤ 100`

📊 Ví dụ

Test case 1

Input:

```
4
PUT user1 Alice
```

```
PUT user2 Bob
GET user1
GET user3
```

Output:

```
[Cache] Put: user1 = Alice
[Cache] Put: user2 = Bob
[Cache] Get: user1 = Alice
[Cache] Get: user3 = null
```

Tags: `singleton`, `double-checked`, `locking`, `volatile`, `optimization`, `pattern`

CodeForge - B05 - Singleton Pattern - Bill Pugh (Recommended)

Độ khó: ★★ Medium

📝 Đề bài

Tạo Singleton với Bill Pugh approach (BEST):

- Class `AppSettings` với:
  - **Private static inner class** `SettingsHolder`:
    - `private static final AppSettings INSTANCE = new AppSettings();`
  - Private constructor
  - `public static AppSettings getInstance()` return SettingsHolder.INSTANCE

Bill Pugh: Lazy + Thread-safe + No synchronization overhead!

Inner class loaded only when getInstance() called (lazy).

◇ Input

- Dòng 1: N (settings)
- N dòng: Key value pairs

◇ Output

- Settings stored

◇ Constraints

- `1 ≤ N ≤ 50`

📊 Ví dụ

Test case 1

Input:

```
3
theme dark
language en
fontSize 14
```

Output:

```
[Settings] theme = dark
[Settings] language = en
[Settings] fontSize = 14
```

Tags: `singleton`, `bill-pugh`, `inner-class`, `best-practice`, `pattern`

CodeForge - B06 - Singleton Pattern - Comparison

Độ khó: ★★ Medium

📝 Đề bài

So sánh các Singleton implementations:

- Eager: Thread-safe, wastes memory if not used
- Lazy: Simple, NOT thread-safe
- Synchronized: Thread-safe, slow (locks every call)
- Double-checked: Thread-safe, complex
- Bill Pugh: BEST - Thread-safe, lazy, no overhead

Demo tất cả 5 implementations và compare.

◇ Input

- Một dòng: Implementation type (EAGER/LAZY/SYNC/DOUBLE/PUGH)

◇ Output

- Characteristics của implementation

◇ Constraints

- N/A

📊 Ví dụ

Test case 1

Input:

```
PUGH
```

Output:

```
Bill Pugh Implementation
Thread-safe: YES
Lazy: YES
Performance: EXCELLENT
Complexity: MEDIUM
Recommended: YES
```

Tags: `singleton`, `comparison`, `best-practice`, `pattern`

CodeForge - B07 - Simple Factory Pattern

Độ khó: ★★ Medium

📝 Đề bài

Tạo Simple Factory:

- Interface `Shape` với `void draw();`
- Classes `Circle`, `Rectangle`, `Triangle` implements Shape
- Class `ShapeFactory` với:
  - `static Shape createShape(String type)`:
    - if "CIRCLE" return new Circle()
    - if "RECTANGLE" return new Rectangle()
    - if "TRIANGLE" return new Triangle()
    - else return null

Simple Factory: Static method creates objects

◇ Input

- Dòng 1: N (shapes)
- N dòng: Shape type

◇ Output

- N drawing messages

◇ Constraints

- `1 ≤ N ≤ 20`

📊 Ví dụ

Test case 1

Input:

```
4
CIRCLE
RECTANGLE
TRIANGLE
CIRCLE
```

Output:

```
Drawing circle
Drawing rectangle
Drawing triangle
Drawing circle
```

Tags: `factory`, `simple`, `creational`, `pattern`

CodeForge - B08 - Factory Method Pattern

Độ khó: ★★★ Hard

📝 Đề bài

Tạo Factory Method pattern:

- Interface `Vehicle` với `void drive();`
- Classes `Car`, `Bike`, `Truck` implements Vehicle
- **Abstract** class `VehicleFactory` với:
  - `abstract Vehicle createVehicle();` (factory method)
  - `void deliverVehicle()`:
    - Vehicle v = createVehicle()
    - v.drive()
- Classes `CarFactory`, `BikeFactory`, `TruckFactory` extends VehicleFactory
  - Implement createVehicle()

Factory Method: Subclasses quyết định object type

◇ Input

- Dòng 1: N (orders)
- N dòng: Factory type (CAR/BIKE/TRUCK)

◇ Output

- N deliveries

◇ Constraints

- `1 ≤ N ≤ 30`

📊 Ví dụ

Test case 1

Input:

```
3
CAR
BIKE
TRUCK
```

Output:

```
Delivering: Driving car
Delivering: Riding bike
Delivering: Driving truck
```

Tags: `factory-method`, `abstract`, `subclass`, `pattern`

CodeForge - B09 - Factory Với Parameters

Độ khó: ★★ Medium

📝 Đề bài

Factory với parameters:

- Interface `Animal` với `void makeSound();`
- Classes `Dog`, `Cat`, `Cow` implements Animal
- Class `AnimalFactory` với:
  - `static Animal createAnimal(String type, String breed)`:
    - Create animal với specific breed
    - Return appropriate type

◇ Input

- Dòng 1: N (animals)
- N dòng: Type breed

◇ Output

- N animals với breeds

◇ Constraints

- `1 ≤ N ≤ 20`

📊 Ví dụ

Test case 1

Input:

```
3
DOG Golden_Retriever
CAT Persian
DOG Bulldog
```

Output:

```
Woof! I'm a Golden_Retriever
Meow! I'm a Persian
Woof! I'm a Bulldog
```

Tags: `factory`, `parameters`, `flexibility`, `pattern`

CodeForge - B10 - Abstract Factory Preview

Độ khó: ★★★ Hard

📝 Đề bài

Preview Abstract Factory (full version Buổi 18):

- Interface `GUIFactory` với:
  - `Button createButton();`
  - `Checkbox createCheckbox();`
- Classes `WindowsFactory`, `MacFactory` implements GUIFactory
  - Create platform-specific components

Abstract Factory: Family of related objects

◇ Input

- Dòng 1: Platform (WINDOWS/MAC)

◇ Output

- Platform-specific components

◇ Constraints

- N/A

📊 Ví dụ

Test case 1

Input:

```
WINDOWS
```

Output:

```
Windows Button created
Windows Checkbox created
```

Test case 2

Input:

```
MAC
```

Output:

```
Mac Button created
Mac Checkbox created
```

Tags: `abstract-factory`, `family`, `preview`, `pattern`

CodeForge - B11 - Factory Với Registry

Độ khó: ★★★ Hard

📝 Đề bài

Factory với dynamic registration:

- Interface `Product` với `String getName();`
- Class `ProductFactory` với:
  - `static Map<String, Product> registry = new HashMap<>();`
  - `static void register(String name, Product prototype)`
  - `static Product create(String name)` return clone of prototype

Registry pattern: Dynamic product registration

◇ Input

- Dòng 1: N (registrations)
- N dòng: Product name
- Dòng N+2: M (creations)
- M dòng: Product name to create

◇ Output

- Created products

◇ Constraints

- `1 ≤ N, M ≤ 20`

📊 Ví dụ

Test case 1

Input:

```
3
Laptop
Phone
Tablet
2
Laptop
Phone
```

Output:

```
Created: Laptop
Created: Phone
```

Tags: `factory`, `registry`, `dynamic`, `pattern`

CodeForge - B12 - Singleton + Factory Combined

Độ khó: ★★★ Hard

📝 Đề bài

Combine Singleton với Factory:

- Class `ConnectionFactory` (Singleton với Bill Pugh) với:
  - Factory methods:
    - `Connection createMySQLConnection()`
    - `Connection createPostgreSQLConnection()`
    - `Connection createMongoConnection()`
  - Maintain connection pool

Pattern combination: Singleton factory

◇ Input

- Dòng 1: N (connections)
- N dòng: DB type (MYSQL/POSTGRES/MONGO)

◇ Output

- Connections created
- All from same factory instance

◇ Constraints

- `1 ≤ N ≤ 30`

📊 Ví dụ

Test case 1

Input:

```
4
MYSQL
POSTGRES
MYSQL
MONGO
```

Output:

```
[Factory Instance #1] MySQL connection created
[Factory Instance #1] PostgreSQL connection created
[Factory Instance #1] MySQL connection created
[Factory Instance #1] MongoDB connection created
Total connections: 4
```

Tags: `singleton`, `factory`, `combined`, `pattern`

CodeForge - B13A - Complete Singleton System - Configuration Manager

Độ khó: ★★★ Hard (Advanced)

📝 Đề bài

Tạo complete configuration system:

- Class `ConfigurationManager` (Bill Pugh Singleton) với:
  - `Map<String, String> configs`
  - `Map<String, List<ConfigListener>> listeners`
  - Methods:
    - `void set(String key, String value)`
    - `String get(String key)`
    - `void addListener(String key, ConfigListener listener)`
    - `void notifyListeners(String key, String oldValue, String newValue)`
- Interface `ConfigListener` với:
  - `void onConfigChanged(String key, String oldValue, String newValue);`
- Classes `LoggingListener`, `CacheInvalidationListener`, `AlertListener` implements ConfigListener

Trong main():

1. Get ConfigurationManager instance
2. Register listeners cho specific keys
3. Update configs
4. Listeners notified automatically
5. Verify singleton (all references same instance)

◇ Input

- Dòng 1: N (listeners)
- N dòng: Listener type, config key
- Dòng N+2: M (config updates)
- M dòng: Key value

◇ Output

- Listener notifications
- Config update log

◇ Constraints

- `1 ≤ N ≤ 10`
- `1 ≤ M ≤ 50`

📊 Ví dụ

Test case 1

Input:

```
3
LOGGING DB_HOST
CACHE API_KEY
ALERT MAX_CONNECTIONS
3
DB_HOST localhost
API_KEY abc123xyz
MAX_CONNECTIONS 100
```

Output:

```
=== Configuration Manager (Singleton) ===
Instance created using Bill Pugh pattern

Registering listeners...
✓ LoggingListener registered for DB_HOST
✓ CacheInvalidationListener registered for API_KEY
✓ AlertListener registered for MAX_CONNECTIONS

Updating configurations...

[Config] DB_HOST: null -> localhost
  → [LoggingListener] Logged change: DB_HOST = localhost

[Config] API_KEY: null -> abc123xyz
  → [CacheInvalidationListener] Cache invalidated for API_KEY

[Config] MAX_CONNECTIONS: null -> 100
  → [AlertListener] Alert: MAX_CONNECTIONS changed to 100

=== Singleton Verification ===
Instance 1: ConfigurationManager@1a2b3c
Instance 2: ConfigurationManager@1a2b3c
Instance 3: ConfigurationManager@1a2b3c
All references identical: true
```

Tags: `singleton`, `observer`, `configuration`, `thread-safe`, `complete`, `advanced`

CodeForge - B14A - Complete Factory System - Plugin Architecture

Độ khó: ★★★ Hard (Advanced)

📝 Đề bài

Tạo extensible plugin system:

- Interface `Plugin` với:
  - `String getName();`
  - `void initialize();`
  - `void execute(String command);`
  - `void shutdown();`
- Classes `SecurityPlugin`, `MonitoringPlugin`, `BackupPlugin` implements Plugin
- **Abstract Factory** `PluginFactory` với:
  - `abstract Plugin createPlugin();`
- **Concrete Factories** `SecurityPluginFactory`, `MonitoringPluginFactory`, `BackupPluginFactory`
- Class `PluginRegistry` (Singleton) với:
  - `Map<String, PluginFactory> factories`
  - `void registerFactory(String name, PluginFactory factory)`
  - `Plugin createPlugin(String name)`
- Class `PluginManager` với:
  - `List<Plugin> activePlugins`
  - `void loadPlugin(String type)`
  - `void executeCommand(String pluginName, String command)`
  - `void shutdownAll()`

Trong main():

1. Get PluginRegistry (Singleton)
2. Register factories
3. Load plugins via PluginManager
4. Execute commands
5. Shutdown gracefully

◇ Input

- Dòng 1: N (plugin types to register)
- N dòng: Plugin type names
- Dòng N+2: M (plugins to load)
- M dòng: Plugin type
- Dòng N+M+3: K (commands)
- K dòng: PluginName command

◇ Output

- Registration log
- Plugin lifecycle
- Command execution

◇ Constraints

- `1 ≤ N ≤ 5`
- `1 ≤ M ≤ 10`
- `1 ≤ K ≤ 20`

📊 Ví dụ

Test case 1

Input:

```
3
SECURITY
MONITORING
BACKUP
3
SECURITY
MONITORING
BACKUP
3
SecurityPlugin scan
MonitoringPlugin check
BackupPlugin start
```

Output:

```
=== Plugin Registry (Singleton) ===

Registering factories...
✓ SecurityPluginFactory registered
✓ MonitoringPluginFactory registered
✓ BackupPluginFactory registered

=== Plugin Manager ===

Loading plugins...
[Factory] Creating SecurityPlugin
  → Initializing SecurityPlugin v1.0
[Factory] Creating MonitoringPlugin
  → Initializing MonitoringPlugin v1.0
[Factory] Creating BackupPlugin
  → Initializing BackupPlugin v1.0

Executing commands...
```

```
[SecurityPlugin] Executing: scan
  → Security scan completed
[MonitoringPlugin] Executing: check
  → System health check completed
[BackupPlugin] Executing: start
  → Backup started

Shutting down...
[SecurityPlugin] Shutting down
[MonitoringPlugin] Shutting down
[BackupPlugin] Shutting down

=== Summary ===
Factories registered: 3
Plugins loaded: 3
Commands executed: 3
Registry instance verified: Singleton
```

Tags: `factory`, `abstract-factory`, `singleton`, `plugin`, `architecture`, `advanced`

CodeForge - B15A - Complete Pattern System - Game Object Factory

Độ khó: ★★★ Hard (Advanced)

📝 Đề bài

Tạo complete game object creation system:

- **Singleton** `GameObjectRegistry` với:
  - Object pool management
  - Factory registration
  - Statistics tracking
- **Factory Method** pattern cho object creation:
  - Abstract `GameObjectFactory`
  - Concrete factories: `EnemyFactory`, `PowerUpFactory`, `ObstacleFactory`
- **Product hierarchy**:
  - Interface `GameObject` với:
    - `String getId();`
    - `void spawn(int x, int y);`
    - `void update();`
    - `void destroy();`
  - Classes: `Enemy`, `PowerUp`, `Obstacle`
- **Object Pool** (Singleton) với:
  - Reuse destroyed objects
  - Lazy initialization
  - Statistics
- **Game Manager** với:
  - Spawn waves of objects
  - Update all active objects
  - Clean up destroyed objects

Trong main():

1. Initialize GameObjectRegistry (Singleton)
2. Register factories
3. Spawn N waves of objects
4. Update lifecycle
5. Show statistics:
   - Objects created
   - Objects reused (from pool)
   - Active objects
   - Destroyed objects

◇ Input

- Dòng 1: N (waves)
- N nhóm:
  - Dòng 1: M (objects in wave)
  - M dòng: Type, position (x, y)

◇ Output

- Object lifecycle logs
- Pool statistics
- Performance metrics

◇ Constraints

- `1 ≤ N ≤ 5`
- `1 ≤ M ≤ 20`

📊 Ví dụ

Test case 1

Input:

```
3
4
ENEMY 10 20
POWERUP 30 40
OBSTACLE 50 60
ENEMY 70 80
3
ENEMY 15 25
POWERUP 35 45
ENEMY 55 65
2
OBSTACLE 75 85
POWERUP 95 105
```

Output:

```
=== Game Object System ===

[Registry] Singleton instance created
[Registry] Registering factories...
  ✓ EnemyFactory
  ✓ PowerUpFactory
  ✓ ObstacleFactory

=== Wave 1 ===
[Factory] Creating Enemy#1 at (10, 20)
[Factory] Creating PowerUp#1 at (30, 40)
```

```
[Factory] Creating Obstacle#1 at (50, 60)
[Factory] Creating Enemy#2 at (70, 80)
Active objects: 4

Updating objects...
Enemy#1 updated
PowerUp#1 updated
Obstacle#1 updated
Enemy#2 updated

=== Wave 2 ===
[Pool] Reusing Enemy#1 at (15, 25)
[Pool] Reusing PowerUp#1 at (35, 45)
[Pool] Reusing Enemy#2 at (55, 65)
Active objects: 7

Updating objects...
Enemy#1 updated (reused)
PowerUp#1 updated (reused)
Obstacle#1 updated
Enemy#2 updated (reused)

=== Wave 3 ===
[Pool] Reusing Obstacle#1 at (75, 85)
[Factory] Creating PowerUp#2 at (95, 105)
Active objects: 9

Updating objects...
Enemy#1 updated (reused)
PowerUp#1 updated (reused)
Obstacle#1 updated (reused)
Enemy#2 updated (reused)
PowerUp#2 updated

=== Final Statistics ===

Object Creation:
  Total created: 6
  From pool (reused): 4
  New allocations: 2
  Pool efficiency: 66.67%

Factory Usage:
  EnemyFactory: 3 objects
  PowerUpFactory: 2 objects
  ObstacleFactory: 1 object

Active Objects: 5
Destroyed Objects: 4

Singleton Verification:
  Registry instance: GameObjectRegistry@abc123
  Pool instance: ObjectPool@def456
  All verified: ✓
```

```
Performance:
  Average spawn time: 0.5ms
  Memory saved (pooling): ~40%
```

Tags: `singleton`, `factory-method`, `object-pool`, `game`, `complete-system`, `capstone`, `advanced`