CODEFORGE - BUỔI 12: STATIC MEMBERS & TOSTRING()

15 BÀI TẬP (12 CORE + 3 ADVANCED)

Kiến thức Buổi 12:

- Static Members (static variables, static methods, static blocks, static import)
- Instance vs Static
- Object Class Methods (toString(), equals() preview, hashCode() preview)

CodeForge - B01 - Static Variable Đếm Objects

Độ khó: ★ Easy
Loại: Competitive

Đề bài

Class `Student` với static variable đếm số lượng:

```
class Student {
    String name;
    static int count = 0;  // Biến static - chung cho tất cả objects

    Student(String name) {
        this.name = name;
        count++;  // Tăng mỗi khi tạo object
    }
}
```

Tạo N students, in số lượng.

◇ Input

- Dòng 1: N
- N dòng: tên

◇ Output

- Số lượng students sau mỗi lần tạo

Ví dụ

Test case 1

Input:

```
5
Alice
Bob
Charlie
David
Eve
```

Output:

```
Tạo student 1: Alice (Tổng: 1)
Tạo student 2: Bob (Tổng: 2)
Tạo student 3: Charlie (Tổng: 3)
Tạo student 4: David (Tổng: 4)
Tạo student 5: Eve (Tổng: 5)

Student.count = 5

Static variable:
✓ count là biến CHUNG cho tất cả objects
✓ Tất cả students đều thấy cùng 1 giá trị count
```

Tags: oop, static, variable, competitive

CodeForge - B02 - Static vs Instance Variables

Độ khó: ★ ★ Medium
Loại: Competitive

Đề bài

So sánh static và instance variables:

```
class BankAccount {
    String accountNumber;        // Instance - mỗi object khác nhau
    double balance;              // Instance
    static double totalBalance;  // Static - chung cho tất cả

    BankAccount(String accNo, double bal) {
        accountNumber = accNo;
        balance = bal;
        totalBalance += bal;  // Cộng vào tổng chung
    }
}
```

Input

- Dòng 1: N (số tài khoản)
- N nhóm 2 dòng: accountNumber, balance

Output

- Thông tin từng tài khoản và tổng chung

Ví dụ

Test case 1

Input:

```
3
ACC001
5000000
ACC002
3000000
ACC003
2000000
```

Output:

```
Tài khoản 1:
Số TK: ACC001 (instance)
Số dư: 5000000đ (instance)
Tổng hệ thống: 5000000đ (static)

Tài khoản 2:
Số TK: ACC002 (instance)
Số dư: 3000000đ (instance)
Tổng hệ thống: 8000000đ (static)

Tài khoản 3:
Số TK: ACC003 (instance)
Số dư: 2000000đ (instance)
Tổng hệ thống: 10000000đ (static)

So sánh:
✓ Instance variables: mỗi object có giá trị riêng
✓ Static variable: CHUNG cho tất cả objects
```

Tags: oop, static, instance, comparison, competitive

CodeForge - B03 - Static Method Tiện Ích

Độ khó: ★★ Medium
Loại: Competitive

Đề bài

Class với static utility methods:

```
class MathUtils {
    static int max(int a, int b) {
        return a > b ? a : b;
    }

    static int min(int a, int b) {
        return a < b ? a : b;
    }

    static double average(int[] arr) {
        int sum = 0;
        for (int num : arr) sum += num;
        return (double) sum / arr.length;
    }
}
```

Gọi static methods mà không cần tạo object.

Input

- Dòng 1: a, b
- Dòng 2: N (kích thước mảng)
- Dòng 3: N số

Output

- Kết quả các phép tính

Ví dụ

Test case 1

Input:

```
15 27
5
10 20 30 40 50
```

Output:

```
MathUtils.max(15, 27) = 27
MathUtils.min(15, 27) = 15

Mảng: [10, 20, 30, 40, 50]
MathUtils.average(arr) = 30.0

Static methods:
✓ Gọi trực tiếp: ClassName.methodName()
✓ Không cần tạo object
✓ Thường dùng cho utility methods
```

Tags: oop, static, methods, utility, competitive

CodeForge - B04 - Static Block Khởi Tạo

Độ khó: ★★ Medium
Loại: Competitive

Đề bài

Class với static block:

```
class Config {
    static String appName;
    static String version;
    static int maxUsers;

    static {
        System.out.println("Static block chạy!");
        appName = "MyApp";
        version = "1.0";
        maxUsers = 100;
    }
}
```

Static block chạy 1 LẦN khi class được load.

Input

```
<choice>
```

- 1: Chỉ truy cập static variables
- 2: Tạo objects

Output

- Thứ tự thực thi

Ví dụ

Test case 1

Input:

```
1
```

Output:

```
Static block chạy!
Khởi tạo: appName = MyApp
Khởi tạo: version = 1.0
Khởi tạo: maxUsers = 100

Truy cập:
Config.appName = MyApp
Config.version = 1.0
Config.maxUsers = 100

Static block:
✓ Chạy 1 LẦN khi class được load
✓ Chạy TRƯỚC constructor
✓ Dùng để khởi tạo static variables
```

Test case 2

Input:

```
2
```

Output:

```
Static block chạy!
Khởi tạo: appName = MyApp

Tạo object 1: Config()
Tạo object 2: Config()

Lưu ý:
✓ Static block CHỈ chạy 1 lần
✓ Constructor chạy mỗi khi new object
```

Tags: oop, static, block, initialization, competitive

CodeForge - B05 - Instance Method vs Static Method

Độ khó: ★★ Medium
Loại: Competitive

Đề bài

So sánh instance và static methods:

```
class Calculator {
    int lastResult;  // Instance variable

    // Instance method - cần object, truy cập được instance variables
    int add(int a, int b) {
        lastResult = a + b;
        return lastResult;
    }

    // Static method - không cần object, KHÔNG truy cập instance variables
    static int multiply(int a, int b) {
        return a * b;
        // lastResult = a * b;  ❌ LỖI! Không truy cập được
    }
}
```

Input

```
<a> <b>
```

Output

- Demo cả 2 loại methods

Ví dụ

Test case 1

Input:

```
5 10
```

Output:

```
Instance method:
Calculator calc = new Calculator();
calc.add(5, 10) = 15
calc.lastResult = 15 ✓ Truy cập được instance variable

Static method:
Calculator.multiply(5, 10) = 50
Không cần object! ✓

Phân biệt:
✓ Instance method: Cần object, truy cập instance variables
✓ Static method: Không cần object, chỉ truy cập static variables

Lỗi thường gặp:
❌ Static method KHÔNG thể truy cập instance variables
❌ Static method KHÔNG thể gọi instance methods
```

Tags: oop, static, instance, methods, comparison, competitive

CodeForge - B06 - toString() Method Cơ Bản

Độ khó: ★ Easy
Loại: Competitive

Đề bài

Override toString() để in object dễ dàng:

```
class Product {
    String name;
    double price;

    Product(String name, double price) {
        this.name = name;
        this.price = price;
    }

    public String toString() {
        return name + " - " + price + "đ";
    }
}
```

◇ Input

- Dòng 1: N
- N nhóm 2 dòng: name, price

◇ Output

- In objects với và không có toString()

Ví dụ

Test case 1

Input:

```
3
Laptop
20000000
Mouse
500000
Keyboard
1500000
```

Output:

```
KHÔNG có toString():
Product@15db9742  ← Địa chỉ bộ nhớ (vô nghĩa)
Product@6d06d69c
Product@7852e922

CÓ toString():
Laptop - 20000000đ
Mouse - 500000đ
Keyboard - 1500000đ

Lợi ích toString():
✓ In object dễ đọc
✓ Tự động gọi khi System.out.println(object)
✓ Debugging thuận tiện
```

Tags: oop, toString, override, competitive

CodeForge - B07 - toString() Với Format Phức Tạp

Độ khó: ★ ★ Medium
Loại: Competitive

Đề bài

toString() với format chi tiết:

```
class Student {
    String id;
    String name;
    int age;
    double gpa;

    public String toString() {
        return String.format("Student[ID=%s, Name=%s, Age=%d, GPA=%.2f]",
                             id, name, age, gpa);
    }
}
```

◇ Input

- Dòng 1: N
- N nhóm 4 dòng: id, name, age, gpa

◇ Output

- Danh sách students với toString()

Ví dụ

Test case 1

Input:

```
3
SV001
Alice
20
3.75
SV002
Bob
22
3.85
SV003
Charlie
```

```
21
3.50
```

Output:

```
Danh sách sinh viên (3):

1. Student[ID=SV001, Name=Alice, Age=20, GPA=3.75]
2. Student[ID=SV002, Name=Bob, Age=22, GPA=3.85]
3. Student[ID=SV003, Name=Charlie, Age=21, GPA=3.50]

Format tips:
✓ Sử dụng String.format() cho format đẹp
✓ %s: String, %d: int, %.2f: double (2 chữ số thập phân)
```

Tags: oop, toString, format, competitive

CodeForge - B08 - Static Counter Trong Nhiều Classes

Độ khó: ★ ★ ★ Hard
Loại: Competitive

Đề bài

Sử dụng static để đếm objects trong nhiều classes:

```
class Student {
    static int count = 0;
    String name;

    Student(String name) {
        this.name = name;
        count++;
    }
}

class Teacher {
    static int count = 0;
    String name;

    Teacher(String name) {
        this.name = name;
        count++;
    }
}
```

Mỗi class có static counter riêng.

Input

- Dòng 1: N (số operations)
- N dòng: S name (Student) hoặc T name (Teacher)

Output

- Số lượng từng loại

Ví dụ

Test case 1

Input:

```
7
S Alice
S Bob
T John
S Charlie
T Jane
T Mike
S David
```

Output:

```
[1] Tạo Student: Alice (Students: 1)
[2] Tạo Student: Bob (Students: 2)
[3] Tạo Teacher: John (Teachers: 1)
[4] Tạo Student: Charlie (Students: 3)
[5] Tạo Teacher: Jane (Teachers: 2)
[6] Tạo Teacher: Mike (Teachers: 3)
[7] Tạo Student: David (Students: 4)

Thống kê:
Student.count = 4
Teacher.count = 3
Tổng: 7 người

Lưu ý:
✓ Mỗi class có static variables RIÊNG
✓ Student.count ≠ Teacher.count
```

Tags: oop, static, counter, multiple-classes, competitive

CodeForge - B09 - equals() Method Cơ Bản (Preview)

Độ khó: ★★ Medium
Loại: Competitive

Đề bài

Override equals() để so sánh objects:

```
class Point {
    int x;
    int y;

    public boolean equals(Object obj) {
        if (obj == null) return false;
        if (!(obj instanceof Point)) return false;
        Point other = (Point) obj;
        return this.x == other.x && this.y == other.y;
    }

    public String toString() {
        return "(" + x + ", " + y + ")";
    }
}
```

Input

```
<x1> <y1>
<x2> <y2>
```

Output

- So sánh với == và equals()

Ví dụ

Test case 1

Input:

```
5 10
5 10
```

Output:

```
Point p1 = (5, 10)
Point p2 = (5, 10)

So sánh reference:
p1 == p2: false  ← Khác objects

So sánh nội dung:
p1.equals(p2): true  ← Cùng x, y

equals() method:
✓ Override để so sánh NỘI DUNG
✓ == so sánh REFERENCES
```

Test case 2

Input:

```
5 10
8 12
```

Output:

```
Point p1 = (5, 10)
Point p2 = (8, 12)

p1.equals(p2): false  ← Khác x hoặc y
```

Tags: oop, equals, override, comparison, competitive

CodeForge - B10 - Static Factory Method

Độ khó: ★ ★ ★ Hard
Loại: Competitive

Đề bài

Sử dụng static method để tạo objects:

```
class Color {
    int red;
    int green;
    int blue;

    private Color(int r, int g, int b) {  // Private constructor
        red = r;
        green = g;
        blue = b;
    }

    // Static factory methods
    static Color createRed() {
        return new Color(255, 0, 0);
    }

    static Color createGreen() {
        return new Color(0, 255, 0);
    }

    static Color createBlue() {
        return new Color(0, 0, 255);
    }

    static Color createCustom(int r, int g, int b) {
        return new Color(r, g, b);
    }

    public String toString() {
        return "RGB(" + red + ", " + green + ", " + blue + ")";
    }
}
```

Input

- Dòng 1: N (số màu)
- N dòng:
  - RED: tạo màu đỏ
  - GREEN: tạo màu xanh lá

- BLUE: tạo màu xanh dương
- CUSTOM r g b: tạo màu tùy chỉnh

◇ Output

- Danh sách màu

📊 Ví dụ

Test case 1

Input:

```
5
RED
GREEN
BLUE
CUSTOM 128 128 0
CUSTOM 255 192 203
```

Output:

```
Màu 1: RGB(255, 0, 0) - Đỏ
Màu 2: RGB(0, 255, 0) - Xanh lá
Màu 3: RGB(0, 0, 255) - Xanh dương
Màu 4: RGB(128, 128, 0) - Tùy chỉnh
Màu 5: RGB(255, 192, 203) - Tùy chỉnh

Static factory methods:
✓ Tên method rõ nghĩa (createRed vs new Color(...))
✓ Có thể trả về subclass
✓ Kiểm soát việc tạo objects
```

Tags: oop, static, factory-method, design-pattern, competitive

CodeForge - B11D - Thiết Kế Singleton Pattern

Độ khó: ★ ★ ★ Hard
Loại: Design

Đề bài

Thiết kế class Singleton (chỉ có 1 instance):

Yêu cầu:

- Private constructor (ngăn new từ bên ngoài)
- Private static instance
- Public static method getInstance()

```
class DatabaseConnection {
    private static DatabaseConnection instance = null;

    private DatabaseConnection() {
        // Private constructor
    }

    public static DatabaseConnection getInstance() {
        if (instance == null) {
            instance = new DatabaseConnection();
        }
        return instance;
    }
}
```

Focus: Đảm bảo chỉ có 1 object được tạo.

Nhiệm vụ

Implement Singleton và demo:

1. Gọi getInstance() nhiều lần
2. Chứng minh chỉ có 1 instance (same reference)

Ví dụ Demo

```
DatabaseConnection db1 = DatabaseConnection.getInstance();
DatabaseConnection db2 = DatabaseConnection.getInstance();
System.out.println(db1 == db2);  // true - cùng object!
```

Tags: oop, singleton, static, design-pattern, design

CodeForge - B12D - Thiết Kế Utility Class

Độ khó: ★ ★ Medium
Loại: Design

Đề bài

Thiết kế class chứa toàn static methods (Utility class):

Yêu cầu:

- Private constructor (ngăn tạo instance)
- Tất cả methods là static
- Không có instance variables

```
class StringUtils {
    private StringUtils() {
        // Ngăn tạo object
    }

    public static boolean isPalindrome(String s) { ... }
    public static String reverse(String s) { ... }
    public static int countVowels(String s) { ... }
    public static String capitalize(String s) { ... }
}
```

Focus: Pure utility class - không cần tạo objects.

Nhiệm vụ

Implement các methods:

- isPalindrome: Kiểm tra chuỗi đối xứng
- reverse: Đảo ngược chuỗi
- countVowels: Đếm nguyên âm (a,e,i,o,u)
- capitalize: Viết hoa chữ cái đầu mỗi từ

Tags: oop, utility-class, static, design

CodeForge - B13D - Thiết Kế Class Với Static Constants

Độ khó: ★ ★ Medium
Loại: Design

Đề bài

Thiết kế class chứa constants:

Yêu cầu:

- Static final constants (không đổi)
- Naming convention: UPPER_CASE
- Grouping constants hợp lý

```
class MathConstants {
    public static final double PI = 3.14159265359;
    public static final double E = 2.71828182846;
    public static final double GOLDEN_RATIO = 1.61803398875;

    private MathConstants() {}  // Ngăn tạo object
}

class GameConfig {
    public static final int MAX_PLAYERS = 4;
    public static final int BOARD_SIZE = 8;
    public static final String GAME_NAME = "Chess";
    public static final String VERSION = "1.0";

    private GameConfig() {}
}
```

Focus: Tổ chức constants khoa học.

Nhiệm vụ

Thiết kế các constants classes:

1. DatabaseConfig (host, port, username, etc.)
2. HttpStatus (200, 404, 500, etc.)
3. ValidationRules (MIN_AGE, MAX_LENGTH, etc.)

Tags: oop, constants, static, final, design

CodeForge - B14D - Thiết Kế toString() Hierarchy

Độ khó: ★ ★ ★ Hard
Loại: Design

Đề bài

Thiết kế toString() cho class hierarchy:

Yêu cầu:

- Base class có toString()
- Subclass override và gọi super.toString()
- Format nhất quán

```
class Person {
    String name;
    int age;

    public String toString() {
        return "Person[name=" + name + ", age=" + age + "]";
    }
}

class Student extends Person {
    String studentId;
    double gpa;

    public String toString() {
        return "Student[" + super.toString() +
               ", id=" + studentId + ", gpa=" + gpa + "]";
    }
}
```

Focus: Reuse parent toString(), maintain consistency.

Nhiệm vụ

Thiết kế hierarchy:

- Shape → Circle, Rectangle
- Vehicle → Car, Bike
- Employee → Manager, Engineer

Mỗi class có toString() hợp lý.

Tags: oop, toString, inheritance, override, design

CodeForge - B15D - Thiết Kế ID Generator (CAPSTONE)

Độ khó: ★ ★ ★ Hard
Loại: Design (CAPSTONE)

Đề bài

CAPSTONE BUỔI 12 - Thiết kế hệ thống tạo ID tự động:

Yêu cầu:

Class IDGenerator (Singleton):

- Private static instance
- Private static int counter
- Private constructor
- Static method getInstance()
- Method generateID(String prefix) → "PREFIX_0001", "PREFIX_0002", ...

Class Entity (sử dụng IDGenerator):

```
class Student {
    private String id;
    private String name;

    public Student(String name) {
        this.id = IDGenerator.getInstance().generateID("STU");
        this.name = name;
    }

    public String toString() {
        return "Student[id=" + id + ", name=" + name + "]";
    }
}

class Product {
    private String id;
    private String name;

    public Product(String name) {
        this.id = IDGenerator.getInstance().generateID("PROD");
        this.name = name;
    }
}
```

Focus: Tổng hợp Singleton + Static + toString()

◇ Nhiệm vụ

1. Implement IDGenerator với Singleton pattern
2. Tạo Student và Product classes sử dụng IDGenerator
3. Demo: Tạo nhiều objects, IDs tự động tăng
4. Chứng minh IDGenerator chỉ có 1 instance

Expected behavior:

```
Student s1 = new Student("Alice");  // id = STU_0001
Student s2 = new Student("Bob");    // id = STU_0002
Product p1 = new Product("Laptop"); // id = PROD_0003
```

Tags: oop, singleton, static, toString, capstone, design