BÁO CÁO SAU BUỔI THỰC HÀNH

Môn học: Lập trình Java
Tên buổi thực hành: Mini Project OOP – Arena of Heroes (Battle Simulation)
Thời gian thực hành: 180 phút
Họ và tên sinh viên: Lê Trung Đông
Mã sinh viên: PTIT-HN-010
Lớp: CNTT5
Nhóm: Nhóm 4

I. NỘI DUNG ĐÃ THỰC HÀNH

1. Trình bày các giải pháp

Xây dựng cấu trúc chương trình

Hệ thống được thiết kế theo mô hình OOP với mục tiêu mô phỏng trận đấu giữa các nhân vật trong đấu trường **Arena of Heroes**.

Kiến trúc bao gồm:

- **GameCharacter (Abstract Class)**
    - Định nghĩa thuộc tính chung: name, hp, attackPower
    - Biến tĩnh static int count để đếm số lượng nhân vật được tạo
    - Phương thức:
        - attack(GameCharacter target) (abstract)
        - takeDamage(int amount)
        - displayInfo()
    - Constructor tăng biến count mỗi khi khởi tạo đối tượng
- **ISkill (Interface)**
    - Định nghĩa hành vi đặc biệt:
        - void useUltimate(GameCharacter target)
- **Warrior (Concrete Class)**

    - Kế thừa GameCharacter
    - Thực thi ISkill
    - Bổ sung thuộc tính armor
- **Mage (Concrete Class)**
    - Kế thừa GameCharacter
    - Thực thi ISkill
    - Bổ sung thuộc tính mana
- **Goblin (Anonymous Class)**
    - Được tạo trực tiếp trong main
    - Kế thừa GameCharacter
    - Không tạo file class riêng

Hệ thống áp dụng đầy đủ các nguyên lý OOP:

- **Kế thừa (Inheritance)**: Warrior và Mage kế thừa GameCharacter
- **Đa hình (Polymorphism)**: Quản lý các nhân vật bằng mảng GameCharacter[]
- **Trừu tượng (Abstraction)**: Sử dụng abstract class và interface
- **Đóng gói (Encapsulation)**: Thuộc tính private, truy cập qua getter/setter

---

Xử lý nghiệp vụ & logic chiến đấu

- Sử dụng mảng đa hình:

GameCharacter[] characters = new GameCharacter[3];

- Duyệt mảng bằng vòng lặp for
- Kiểm tra phần tử khác null trước khi xử lý
- Thực hiện mô phỏng:
    - Các nhân vật tấn công lẫn nhau
    - Gọi useUltimate() cho Warrior và Mage
- In trạng thái cuối trận:

    - HP
    - Mana (nếu có)
    - Armor (nếu có)
- Hiển thị tổng số nhân vật bằng GameCharacter.count

---

2. Liệt kê các câu hỏi phản biện

Câu hỏi 1:

Tại sao cần sử dụng Abstract Class thay vì chỉ dùng Interface?

Trả lời:
Abstract class cho phép định nghĩa trạng thái chung (name, hp, attackPower) và logic xử lý mặc định (takeDamage), trong khi interface chỉ mô tả hành vi. Vì nhân vật có bản chất chung nên cần abstract class để tránh lặp code.

Câu hỏi 2:

Vì sao sử dụng biến static count trong GameCharacter?

Trả lời:
Biến static thuộc về lớp thay vì từng đối tượng. Nhờ đó hệ thống có thể đếm chính xác tổng số nhân vật đã được tạo mà không phụ thuộc vào từng instance riêng lẻ.

Câu hỏi 3:

Tại sao Goblin được tạo bằng Anonymous Class?

Trả lời:
Goblin chỉ phục vụ mục đích minh họa cơ chế đa hình và override động. Việc dùng Anonymous Class giúp:

- Giảm số lượng file
- Thể hiện rõ khả năng kế thừa tại runtime
- Phù hợp với đối tượng dùng một lần

---

3. Thực hành triển khai code

Thiết kế lớp GameCharacter

- Khai báo thuộc tính private
- Constructor tăng biến count
- Phương thức takeDamage():

HP mới = HP cũ − sát thương
Nếu HP <= 0 → in "<Tên> đã bị hạ gục"

Thiết kế lớp Warrior

- Thuộc tính: armor
- attack():

target.takeDamage(attackPower);

- takeDamage():

damage thực tế = damage − armor
nếu < 0 → = 0

- useUltimate():
    - Gây sát thương gấp đôi
    - Tự mất 10% HP hiện tại

Thiết kế lớp Mage

- Thuộc tính: mana
- attack():
    - Nếu mana >= 5 → trừ 5 mana
    - Nếu mana < 5 → sát thương = attackPower / 2
- useUltimate():
    - Điều kiện mana >= 50
    - Trừ 50 mana

- Gây sát thương lớn (attackPower * 3)

Anonymous Class – Goblin

```
GameCharacter goblin = new GameCharacter("Goblin", 100, 10) {
  @Override
  public void attack(GameCharacter target) {
    System.out.println("Goblin cắn trộm...");
    target.takeDamage(10);
  }
};
```

Đặc điểm:

- Không sử dụng attackPower
- Gây sát thương cố định 10

II. CÔNG VIỆC CÁC EM ĐÃ LÀM

1. Công việc cá nhân

- Hoàn thành:
  - Abstract Class GameCharacter
  - Interface ISkill
  - Warrior và Mage
  - Anonymous Class Goblin
- Cài đặt logic:
  - Trừ giáp của Warrior
  - Trừ mana của Mage
  - Tự động hạ gục khi HP ≤ 0
- Sử dụng đa hình quản lý danh sách nhân vật
- In tổng số nhân vật bằng biến static

---

2. Công việc nhóm

- Thảo luận kịch bản mô phỏng trận đấu
- Chuẩn hóa format in kết quả theo dạng:

=== ARENA OF HEROES ===

- Phân tích khả năng mở rộng:
    - Thêm lớp Archer mà không sửa vòng lặp
- Đảm bảo hệ thống tuân thủ nguyên tắc Open–Closed

III. KẾT QUẢ ĐẠT ĐƯỢC

Sau buổi thực hành, em đã:

- Hiểu rõ cách kết hợp Abstract Class và Interface
- Áp dụng đa hình trong mảng đối tượng
- Nắm vững cơ chế override phương thức
- Thành thạo Anonymous Class
- Biết sử dụng static để quản lý dữ liệu cấp lớp
- Xây dựng thành công hệ thống mô phỏng trận đấu hoàn chỉnh

Hệ thống chạy đúng yêu cầu SRS và hiển thị kết quả tương tự mẫu đề bài.

IV. KHÓ KHĂN VÀ VẤN ĐỀ GẶP PHẢI

Khó khăn

- Phân biệt vai trò giữa abstract class và interface
- Xử lý phần giảm giáp của Warrior sao cho không âm
- Kiểm soát mana của Mage tránh âm

Lỗi gặp phải

- NullPointerException khi chưa kiểm tra phần tử null trong mảng
- Lỗi logic khi mana âm

- Sai sót khi tính 10% HP hiện tại của Warrior

Cách khắc phục

- Kiểm tra null trước khi gọi phương thức
- Validate giá trị mana trước khi trừ
- Tính toán theo công thức:

```
hp = hp - (int)(hp * 0.1);
```

V. KINH NGHIỆM RÚT RA

- Luôn thiết kế sơ đồ class trước khi code
- Xác định rõ:
    - Bản chất → abstract class
    - Hành vi → interface
- Tận dụng đa hình để tránh if–else phân loại đối tượng
- Kiểm tra logic sát thương và trạng thái nhân vật sau mỗi lượt
- Thiết kế hệ thống theo hướng mở rộng (Extensible Design)

VI. ĐỀ XUẤT / KIẾN NGHỊ

- Có thể mở rộng hệ thống:
    - Thêm nhân vật Archer, Assassin
    - Thêm cơ chế Critical Damage
    - Thêm hệ thống turn-based đầy đủ
- Chuyển từ mảng sang ArrayList để linh hoạt số lượng nhân vật
- Phát triển thêm giao diện GUI hoặc tích hợp lưu dữ liệu

VII. KẾT LUẬN

Tự đánh giá:
Hoàn thành đầy đủ yêu cầu của mini-project. Áp dụng chính xác các nguyên lý OOP: Kế thừa, Đa hình, Trừu tượng và Đóng gói.