# Bài tập lớn Lập trình Hướng đối tượng
# Phát triển game Arkanoid

# 1. Giới thiệu

Bài tập lớn này nhằm mục đích củng cố kiến thức về lập trình hướng đối tượng (OOP) thông qua việc phát triển một phiên bản đơn giản của trò chơi kinh điển Arkanoid. Sinh viên sẽ áp dụng các nguyên tắc cơ bản của OOP như đóng gói, kế thừa, đa hình và trừu tượng hóa để xây dựng một ứng dụng có cấu trúc rõ ràng, dễ bảo trì và mở rộng.

Chơi thử game tai đây hoặc tham khảo video gameplay tai đây.

# 2. Mục tiêu

- Nắm vững và áp dụng các nguyên tắc OOP vào thiết kế và triển khai phần mềm.
- Phát triển kỹ năng phân tích, thiết kế hệ thống và giải quyết vấn đề.
- Xây dựng một ứng dụng game hoàn chỉnh từ phiên bản Terminal đến giao diện đồ họa.
- Rèn luyện kỹ năng làm việc nhóm và quản lý dự án phần mềm.

# 3. Yêu cầu chung

Trong quá trình thực hiện, sinh viên được phép bổ sung thêm các lớp và hàm cần thiết, cũng như tùy chỉnh cấu trúc chương trình. Có thể sử dụng thêm các thư viện Java bên ngoài nếu cần. Khuyến khích sinh viên sử dụng các công nghệ, thư viện nâng cao để trau dồi kỹ năng.

# 4. Các phiên bản triển khai

## 4.1. Phiên bản Terminal cơ bản

Phiên bản này tập trung vào việc triển khai các chức năng cơ bản của game Arkanoid thông qua giao diện terminal.

### 4.1.1. Chức năng chính

- **Quản lý trạng thái game:** Bắt đầu game, kết thúc game, quản lý điểm số, số mạng, cấp độ,...
- **Điều khiển Paddle:** Di chuyển thanh đỡ (Paddle) sang trái/phải.
- **Di chuyển Ball:** Xử lý va chạm của bóng với Paddle, tường, và các khối gạch.
- **Phá hủy Brick:** Khi bóng va chạm, khối gạch sẽ bị phá hủy (hoặc giảm độ bền).
- **Xử lý Power-up:** Thu thập và áp dụng các hiệu ứng của Power-up (ví dụ: tăng kích thước Paddle, tăng tốc độ bóng, bóng xuyên phá).
- **Xử lý các trường hợp lỗi:** Ví dụ: bóng rơi khỏi màn hình (mất mạng), không thể di chuyển Paddle ra ngoài biên.

- **Giao diện dòng lệnh đơn giản:** Hiển thị trạng thái game, vị trí các đối tượng (Paddle, Ball, Brick) trên màn hình console.

#### 4.1.2. Cài đặt phiên bản Terminal

- Xây dựng giao diện menu tương tác với người dùng (ví dụ: Bắt đầu game, Tạm dừng, Thoát).
- Người dùng sử dụng các phím mũi tên tương ứng để điều khiển hoặc thực hiện hành động.
- Xử lý các trường hợp nhập liệu không hợp lệ.

### 4.2. Phiên bản giao diện đồ họa (GUI)

Tiếp tục phát triển ứng dụng với giao diện đồ họa để nâng cao trải nghiệm người dùng.

#### 4.2.1. Xây dựng giao diện người dùng

- Sử dụng Swing hoặc JavaFX để tạo giao diện đồ họa cho game.
- Giao diện bao gồm:
    - **Màn hình gameplay:** Hiển thị Paddle, Ball, Bricks, Power-ups.
    - **Thông tin game:** Hiển thị điểm số, số mạng, cấp độ hiện tại.
    - **Menu điều khiển:** Các nút hoặc menu để bắt đầu, tạm dừng, thoát game.

#### 4.2.2. Tích hợp hiệu ứng âm thanh và hình ảnh

- Thêm hiệu ứng âm thanh cho các sự kiện trong game (va chạm, phá gạch, thu Power-up).
- Sử dụng hình ảnh (sprites) cho các đối tượng trong game để tăng tính thẩm mỹ.

#### 4.2.3. Sử dụng đa luồng

- Thực hiện các tác vụ nặng như cập nhật trạng thái game, xử lý va chạm trong luồng riêng để đảm bảo giao diện người dùng không bị treo.
- Đảm bảo game chạy mượt mà và phản hồi nhanh.

### 4.3. Cải tiến (Tùy chọn)

Các yêu cầu nâng cao, khuyến khích sự sáng tạo và ứng dụng kiến thức.

#### 4.3.1. Chức năng tự sáng tạo

- **Hệ thống cấp độ:** Thiết kế nhiều cấp độ với bố cục gạch và độ khó khác nhau.
- **Các loại Brick đặc biệt:** Thêm các loại gạch có thuộc tính khác nhau (ví dụ: gạch không phá hủy, gạch cần nhiều lần va chạm, gạch phát nổ).
- **Power-up đa dạng:** Bổ sung thêm các loại Power-up mới với hiệu ứng độc đáo (ví dụ: bắn laser, tạo thêm bóng, nam châm hút bóng).

- **Multi-player:** Cho phép nhiều người chơi trong một ván game.
- **Save/Load game:** Cho phép người chơi lưu lại trạng thái game và tiếp tục chơi.
- **Bảng xếp hạng:** Lưu trữ và hiển thị điểm số cao nhất của người chơi.

#### 4.3.2. Cải thiện giao diện người dùng

- Giao diện, âm thanh, hình ảnh đặc sắc, sử dụng các nguyên tắc thiết kế UI/UX.
- Cá nhân hoá giao diện theo từng người chơi (ví dụ: chọn màu Paddle, hình dạng bóng, animation).

# 5. Thiết kế hướng đối tượng (OOP)

## 5.1. Xác định các lớp chính và triển khai kế thừa

Sinh viên cần xác định các lớp cần thiết để xây dựng game Arkanoid, bao gồm các lớp cơ sở và các lớp con kế thừa. Dưới đây là một gợi ý về cấu trúc lớp:

- **`GameObject` (Abstract Class/Interface):** Lớp cơ sở cho tất cả các đối tượng trong game có vị trí, kích thước và có thể được vẽ trên màn hình.
  - Thuộc tính: `x`, `y`, `width`, `height`.
  - Phương thức: `update()`, `render()`.

- **`MovableObject` (Abstract Class, kế thừa từ `GameObject`):** Lớp cơ sở cho các đối tượng có thể di chuyển.
  - Thuộc tính: `dx`, `dy` (tốc độ di chuyển theo trục x, y).
  - Phương thức: `move()`.

- **`Paddle` (Kế thừa từ `MovableObject`):** Thanh đỡ mà người chơi điều khiển.
  - Thuộc tính: `speed`, `currentPowerUp`.
  - Phương thức: `moveLeft()`, `moveRight()`, `applyPowerUp()`.

- **`Ball` (Kế thừa từ `MovableObject`):** Quả bóng trong game.
  - Thuộc tính: `speed`, `directionX`, `directionY`.
  - Phương thức: `bounceOff(GameObject other)`, `checkCollision(GameObject other)`.

- **`Brick` (Kế thừa từ `GameObject`):** Các khối gạch cần phá hủy.
  - Thuộc tính: `hitPoints` (số lần va chạm để phá hủy), `type` (loại gạch).

    - Phương thức: `takeHit()`, `isDestroyed()`.

- **`NormalBrick` (Kế thừa từ `Brick`):** Gạch thông thường, bị phá hủy sau một lần va chạm.

- **`StrongBrick` (Kế thừa từ `Brick`):** Gạch cứng, cần nhiều hơn một lần va chạm để phá hủy.

- **`PowerUp` (Kế thừa từ `GameObject`):** Các vật phẩm tăng sức mạnh rơi ra khi phá gạch.
    - Thuộc tính: `type` (loại Power-up), `duration`.
    - Phương thức: `applyEffect(Paddle paddle)`, `removeEffect(Paddle paddle)`.

- **`ExpandPaddlePowerUp` (Kế thừa từ `PowerUp`):** Tăng kích thước Paddle.

- **`FastBallPowerUp` (Kế thừa từ `PowerUp`):** Tăng tốc độ bóng.

- **`GameManager` (hoặc `GameEngine`):** Lớp quản lý toàn bộ logic game.
    - Thuộc tính: `paddle`, `ball`, `bricks` (danh sách các khối gạch), `powerUps` (danh sách các Power-up), `score`, `lives`, `gameState`.
    - Phương thức: `startGame()`, `updateGame()`, `handleInput()`, `checkCollisions()`, `gameOver()`.

- **`Renderer` (hoặc `GameView`):** Lớp chịu trách nhiệm vẽ các đối tượng để hiển thị trên màn hình.
    - Phương thức: `draw(GameObject obj)`.

## 5.2. Áp dụng các nguyên tắc OOP

- **Đóng gói (Encapsulation):** Sử dụng các mức truy cập (private, public, protected) phù hợp cho các thuộc tính và phương thức của mỗi lớp. Ví dụ: thuộc tính `x, y` của `GameObject` nên là `private` và cung cấp các phương thức `public` để truy cập hoặc thay đổi.

- **Kế thừa (Inheritance):** Sử dụng kế thừa để tái sử dụng mã và tạo ra một hệ thống phân cấp lớp rõ ràng. Ví dụ: `Paddle`, `Ball`, `Brick` kế thừa từ `GameObject` hoặc `MovableObject`.
- **Đa hình (Polymorphism):** Các phương thức có thể hoạt động khác nhau tùy theo lớp con. Ví dụ: phương thức `render()` trong `GameObject` có thể được ghi đè (override) trong các lớp con như `Paddle`, `Ball`, `Brick` để vẽ các đối tượng này theo cách riêng của chúng. Tương tự, phương thức `applyEffect()` của `PowerUp` sẽ có hành vi khác nhau ở các lớp con như `ExpandPaddlePowerUp` hay `FastBallPowerUp`.
- **Trừu tượng hóa (Abstraction):** Đơn giản hóa ứng dụng bằng cách ẩn đi các chi tiết không cần thiết. Sử dụng các lớp trừu tượng (`GameObject`, `MovableObject`) và interface để định nghĩa các hành vi chung mà không cần quan tâm đến chi tiết triển khai cụ thể. Ví dụ: `GameObject` định nghĩa `update()` và `render()` mà không cần biết `Paddle` hay `Ball` sẽ cập nhật và vẽ như thế nào.

## 6. Chất lượng code và quy tắc lập trình

- **Đặt tên:** Đặt tên biến, phương thức, và lớp đúng và dễ hiểu, tuân thủ các quy tắc đặt tên trong lập trình Java (camelCase cho biến/phương thức, PascalCase cho lớp) và sử dụng tiếng Anh.
- **Coding Convention:** Tuân thủ chuẩn code về thụt lề, khoảng trắng và cấu trúc mã (ví dụ: theo Google Java Style Guide).
- **Comments:** Thêm chú thích (comments) cho các đoạn code phức tạp hoặc các phần logic quan trọng.
- **Unit Test (Tùy chọn):** Cài đặt Unit Test bằng JUnit cho các thành phần logic quan trọng của game (ví dụ: kiểm tra va chạm, cập nhật điểm số).
- **Design Pattern (Tùy chọn):** Áp dụng các Design Pattern phù hợp (ví dụ: Singleton cho `GameManager`, Factory Method cho việc tạo `Brick` hoặc `PowerUp`).

## 7. Trình bày và hỏi đáp

- Trình bày các tính năng chính, kiến trúc hệ thống và các kỹ thuật đã sử dụng.
- Sẵn sàng trả lời các câu hỏi về thiết kế, mã nguồn và quá trình phát triển.

## 8. Chấm điểm

| Yêu cầu | Điểm | Mức |
|---|---|---|
| **Thiết kế lớp và cây kế thừa** | | |
| Xác định các lớp chính và triển khai kế thừa giữa các lớp | 0.5 | Bắt buộc |
| Sử dụng đúng các nguyên tắc OOP | 0.5 | Bắt buộc |
| **Chức năng chính của ứng dụng** | | |
| Quản lý trạng thái game, điều khiển Paddle, di chuyển Ball, phá hủy Brick | 1 | Bắt buộc |
| Xử lý Power-up, xử lý các trường hợp lỗi | 1 | Bắt buộc |
| Giao diện người dùng (dòng lệnh hoặc GUI) | 0.5 | Bắt buộc |
| Tích hợp hiệu ứng âm thanh và hình ảnh (GUI) | 0.5 | Tự chọn |
| Sử dụng đa luồng để cải thiện trải nghiệm người dùng (GUI) | 0.5 | Tự chọn |
| **Chức năng tự sáng tạo** | | |
| Hệ thống cấp độ, các loại Brick đặc biệt, Power-up đa dạng | 0.5 | Tự chọn |
| Lưu/Tải game, Bảng xếp hạng | 0.5 | Tự chọn |
| **Chất lượng code và quy tắc lập trình** | | |
| Đặt tên biến, phương thức, và lớp đúng và dễ hiểu | 0.5 | Bắt buộc |
| Sử dụng các nguyên tắc coding convention | 0.5 | Bắt buộc |
| Commit code thường xuyên và đúng quy tắc | 0.5 | Bắt buộc |
| Cài đặt Unit Test bằng JUnit | 0.5 | Tự chọn |
| Sử dụng Design Pattern vào ứng dụng | 0.5 | Tự chọn |
| **Trình bày + hỏi đáp** | 3 | Bắt buộc |
| **Tổng điểm** | **10 + 1** | |