TRƯỜNG ĐẠI HỌC SÀI GÒN
KHOA CÔNG NGHỆ THÔNG TIN

Báo cáo học phần Phương pháp lập trình hướng đối tượng

TRÒ CHƠI NÔNG TRẠI

[Nhóm 08] sinh viên thực hiện:
Trần Hoàng Lan Phương – 3124411242
Nguyễn Hoài Quân –3124411249
Huỳnh Thị Thủy Tiên – 3124411308
Trần Hạnh Vy – 3124411355
Giảng viên hướng dẫn: : Nguyễn Minh Hải

Thành phố Hồ Chí Minh, tháng 11 năm 2025

LỜI CAM ĐOAN

Chúng tôi xin cam đoan, bài báo cáo là sản phẩm do nhóm chúng tôi thực hiện tuân thủ các nguyên tắc, kết cấu của một bài báo cáo. Là sản phẩm của tập thể nhóm và không có bất kì sự gian lận hay sao chép nào. Các cơ sở lý luận và kiến thức được trình bày trong bài tiểu luận là hoàn toàn trung thực, có nguồn gốc rõ ràng, được chọn lọc từ nhiều nguồn tài liệu đính kèm chi tiết và hợp lệ.

Nhóm chúng tôi xin hoàn toàn chịu trách nhiệm trước nhà trường và giảng viên hướng dẫn nếu có bất kỳ hành vi gian lận hay thông tin không trung thực nào liên quan đến nội dung của bài báo cáo này.

MỤC LỤC

DANH MỤC CÁC KÝ HIỆU VÀ CHỮ VIẾT TẮT

Viết tắt	Đầy đủ	Nghĩa
Abstract	Abstract class	Lớp trừu tượng
CLI	Command Line Interface	Giao diện Dòng lệnh (Console/Terminal)
Console/Terminal		Giao diện nhập/xuất lệnh
Controller		Lớp Điều khiển (Trong mô hình thiết kế)
CRUD	Create, Read, Update, Delete	Bộ bốn thao tác cơ bản trong quản lý dữ liệu
CSS	Cascading Style Sheets	Ngôn ngữ định kiểu cho HTML
Data Managers		Các lớp quản lý dữ liệu (Kho, Chuong, Vuon)
File		Tệp tin
game loop		Vòng lặp chính của trò chơi
game_save.txt		Tên tệp tin lưu trữ trạng thái trò chơi
GameData		Lớp chứa dữ liệu thô phục vụ việc tải game
GameMenu		Lớp quản lý vòng lặp và menu chính của trò chơi

HTML	HyperText Markup Language	Ngôn ngữ đánh dấu siêu văn bản
I/O	Input/Output	Đầu vào/Đầu ra (Thao tác đọc/ghi dữ liệu)
Interface		Giao diện (trong lập trình hướng đối tượng)
JavaScript		Ngôn ngữ lập trình web
OOP	Object-Oriented Programming	Lập trình Hướng đối tượng
Stacking		Cơ chế xếp chồng (gộp số lượng vật phẩm trong kho)
UML	Unified Modeling Language	Ngôn ngữ mô hình hóa thống nhất

DANH MỤC CÁC HÌNH VẼ

DANH MỤC CÁC BIỂU ĐỒ, BẢNG BIỂU

LỜI MỞ ĐẦU

Giới thiệu đề tài

Các đề tài bài tập lớn được gợi ý đều được thiết kế nhằm mục tiêu giúp sinh viên củng cố và vận dụng kiến thức lý thuyết đã học vào thực tiễn. Yêu cầu cốt lõi của tất cả các đề tài là sinh viên phải tận dụng tối đa những nguyên lý nền tảng của lập trình hướng đối tượng (OOP), bao gồm: tính đóng gói, tính kế thừa, tính đa hình và tính trừu tượng. Nhận thấy đề tài "Trò chơi nông trại" không chỉ đáp ứng đầy đủ các yêu cầu về mặt học thuật mà còn mang tính ứng dụng thực tiễn và gần gũi với người dùng, nhóm 8 quyết định lựa chọn đề tài này để triển khai dự án, nhằm tối ưu hóa việc vận dụng các nguyên lý OOP đã học.

Tóm tắt mục tiêu:

- Thiết kế mô hình đối tượng: Xây dựng một hệ thống lớp có tính kế thừa rõ ràng (*Bo*, *Ga* kế thừa từ *DongVat*, *DongVat* kế thừa từ *SinhVat*).
- Áp dụng tính đa hình: Cho phép các đối tượng khác nhau (*Bo* và *Ga*) có cùng một hành động (*choSanPham()*) nhưng cho ra kết quả khác nhau (*Sua* hoặc *Trung*).
- Sử dụng Interface: Định nghĩa các hành vi chung như *NangCap* (cho phép nâng cấp *Chuong, Vuon*) và *MuaBan* (cho *CuaHang*).
- Xây dựng ứng dụng hoàn chỉnh: Tích hợp các module quản lý (*Chuong*, *Vuon, Kho*), hệ thống nhiệm vụ và vòng lặp trò chơi (*GameMenu*) để tạo ra một ứng dụng có thể hoạt động.

ĐÓNG GÓP

Thành viên	Phương (3124411242)	Quân (3124411249)	Tiên (3124411308)	Vy (3124411355)
Phần trăm đóng góp	27%	20%	29%	25%

CHƯƠNG 1. PHÂN TÍCH VÀ MÔ HÌNH HÓA ĐỐI TƯỢNG

1.1. Phân tích yêu cầu và luật chơi cốt lõi

Dự án mô phỏng "Trò chơi nông trại" trên giao diện dòng lệnh (CLI) được xây dựng dựa trên các luật chơi và yêu cầu cốt lõi, định hình cấu trúc lớp và mối quan hệ trong toàn bộ hệ thống.

- Thực thể cốt lõi

Khái niệm/ Thực thể	**Luật chơi/ Giả định cốt lõi**	**Lớp thực thi**
Người chơi	Người chơi (*NguoiChoi*) khởi đầu với 1000.0 Tiền. Người chơi quản lý trực tiếp Tien, *Kho*, *Chuong*, *Vuon* và danh sách *NhiemVu*.	*NguoiChoi.java*
Sinh vật (động vật, thực vật)	*SinhVat* có các thuộc tính chung (*maso, ten, trangthai, tuoi*) và các hành vi trừu tượng (*phattrien(), hienthitrangthai()*)	*SinhVat.java* (abstract)
Động vật	Kế thừa từ *SinhVat*, có thêm *LoaiThucAn*. Các động vật cụ thể *Bo, Ga, Heo* triển khai hành vi riêng (*keu(), an()*) và triển khai giao diện interface *Thuhoach* để có thể tạo ra *Sanpham* qua phương thức *chosanpham()*	*DongVat.java, Bo.java*, *Ga.java*, *Heo.java* *(implements Thuhoach)*
Thực vật	Lớp *Thucvat* kế thừa từ *SinhVat*, có thêm *giaiDoanPhatTrien* và trạng thái *daTuoiNuoc*. Các cây cụ thể (*Lua*,	*Thucvat.java, Ngo.java, Lua.java, Mamxoi.java* *(implements Thuhoach)*

	Ngo, *Mamxoi*) triển khai phương thức *tuoiNuoc()* và triển khai giao diện *Thuhoach* để tạo ra *Sanpham*	
Sản phẩm	Sản phẩm (*Sanpham)* là kết quả thu hoạch, có *TenSP*, *giaBan* và *soLuong*. *Kho* quản lý *Sanpham* theo cơ chế stack (gộp số lượng).	*Sanpham.java* (abstract), *Kho.java*

Bảng 1.1. Phân tích thực thể cốt lõi

– Cơ chế quản lý nông trại

Chức năng	**Quy tắc đã triển khai**	**Lớp/ Interface liên quan**
Lưu trữ sinh vật	Giới hạn số lượng sinh vật được lưu trữ, *Chuong* (*SucChuaToiDa* = 5) và *Vuon* (*SucChuaToiDa* = 10)	*Chuong.java*, *Vuon.java*
Hành động chung	Chức năng *HangDong* (case 6 trong menu) cho phép gọi đồng loạt *choTatCaAn()* class *Chuong*, *tuoiNuocTatCa()* class *Vuon* và *tiengKeuDongLoat()* class *Chuong* gọi *keu()* của từng *DongVat* theo nguyên lý đa hình	*GameMenu.java, Chuong.java, DongVat.java, Bo.java, Ga.java, Heo.java, Vuon.java*
Thu hoạch	Gọi phương thức *thuHoachTatCa()*. Các *SinhVat* phải là *Thuhoach* và cung cấp *Sanpham* bằng phương thức *chosanpham()*. Sản phẩm được tự động gộp (bằng stack) khi thêm vào *Kho*	*Thuhoach.java, Chuong.java, Vuon.java, Kho.java*
Nâng cấp	*Chuong* và *Vuon* triển khai nâng cấp. Mỗi lần *nangCap()* tăng	*NangCap.java, Chuong.java, Vuon.java, NguoiChoi.java*

	SucChuaToiDa (chuồng +3, vườn +5) và tăng *ChiPhiNangCap* x 1,5. Yêu cầu người chơi phải có đủ tiền để thực hiện	
Cửa hàng	Triển khai *MuaBan, banSanPham()*, lấy *giaBan* từ *Sanpham* trong *Kho* để tính tiền nhận được. *muaVatPham():* trừ tiền và thêm *SinhVa*t mới vào *Chuong* hoặc *Kho*	MuaBan.java, CuaHang.java

Bảng 1.2. Cơ chế vận hành và quản lý chức năng nông trại

– Cơ chế nhiệm vụ

Chức năng	**Quy tắc đã triển khai**	**Lớp/ Interface liên quan**
Phân loại	Nhiệm vụ được phân loại thành *NhiemVuNangCap* (kiểm tra *capDo* công trình) và *NhiemVuThuHoach* (kiểm tra *soLuong* sản phẩm trong *Kho*)	*Nhiemvu.java (abstract), NhiemVuNangCap.java, NhiemVuThuHoach.java*
Kiểm tra tự động	Hàm *kiemTraTatCaNhiemVu()* được gọi tự động sau mỗi hành động chính của người chơi để kiểm tra và hoàn thiện nhiệm vụ nếu đủ điều kiện đặt ra.	*NguoiChoi.java*
Trừu tượng hóa	*NhiemVu* là lớp trừu tượng, buộc các lớp con phải triển khai logic kiểm tra cụ thể thông qua phương thức trừu tượng *kiemTraHoanThanh* lớp *NguoiChoi.*	*Nhiemvu.java*

Bảng 1.3. Cơ chế và cấu trúc hệ thống nhiệm vụ

– Cơ chế dữ liệu

Chức năng	Quy tắc đã triển khai	Lớp/ Interface liên quan
Lưu và tải Game	Hệ thống sẽ duyệt qua các đối tượng *NguoiChoi*, *Chuong*, *Vuon, Kho*, lấy ra các thuộc tính quan trọng (*Tien*, c*apDo*, *maso*,…) và ghi chúng thành các dòng văn bản vào file game_save.txt	*XuLyFile.java, GameData.java*
Tái tạo đối tượng	Quá trình *taiGame* sử dụng *LuuQuaTrinh* để tái tạo đối tượng cụ thể (*Bo* hay *Ga*) dựa trên dữ liệu đã lưu (*maso, ten, tuoi*)	*LuuQuaTrinh.java*

Bảng 1.4. Cơ chế lưu trữ và tái tạo dữ liệu

1.2. Xác định và mô tả chi tiết các lớp

Hệ thống được mô hình hóa dựa trên nguyên tắc phân tách trách nhiệm, chia thành ba nhóm lớp chính. Sự phân chia này đảm bảo tính độc lập, dễ bảo trì và dễ mở rộng của mã nguồn

1.2.1. Nhóm lớp cơ sở và sinh vật (hệ thống kế thừa và đa hình)

Nhóm này thiết lập nền tảng phân cấp cho các thực thể vật lý trong game, sử dụng chủ yếu nguyên lý kế thừa để tái sử dụng mã và đa hình để tùy biến hành vi

Lớp	Vai trò	Lớp con
SinhVat (abstract)	Lớp cha cơ sở, định nghĩa trạng thái và hành vi chung cho mọi sinh vật (*maso, ten, tuoi*). Là điểm khởi đầu của chuỗi kế thừa	*DongVat.java, Thucvat.java*
DongVat (abstract)	Kế thừa *SinhVat*. Trừu tượng hóa các hành vi đặc thù (*keu(),an()*) và là đối tượng triển khai giao diện *Thuhoach*	*Bo.java, Ga.java, Heo.java*

Thucvat (abstract)	Kế thừa *SinhVat*, quản lý trạng thái vòng đời (*giaiDoanPhatTrien*, *daTuoiNuoc*) và hành vi *tuoiNuoc()*. Cũng triển khai *Thuhoach*	*Lua.java, Ngo.java, Mamxoi.java*
Sanpham (abstract)	Lớp cơ sở cho các sản phẩm, quản lý giá trị và số lượng. Là nền tảng cho cơ chế stacking (gộp số lượng) trong *Kho*	*Sua.java, Thit.java, Trung java, Gao.java, BapNgo.java, Traimamxoi.java*

Bảng 1.5. Mô hình lớp cơ sở và kế thừa

1.2.2. Nhóm lớp công trình và quản lý tài nguyên

Nhóm này quản lý các container (vật chứa) và logic kinh tế cốt lõi, sử dụng nguyên lý hợp thành để quản lý các *SinhVat* và Interface để định nghĩa khả năng

- *Chuong* và *Vuon*: Đóng vai trò là các container quản lý danh sách *DongVat* và *Thucva*t thông qua mối quan hệ hợp thành. Cả hai đều triển khai *NangCap*, cho phép mô hình hóa việc tăng sức chứa và chi phí theo cấp độ. Đặc biệt, logic *thuHoachTatCa()* sử dụng cấu trúc Map để gộp (stack) sản phẩm thu hoạch, tối ưu hóa quá trình thêm vào *Kho*
- *Kho:* Đóng vai trò là quản lý tồn kho, lớp này chịu trách nhiệm chính trong việc thực thi quy tắc stacking số lượng sản phẩm (*Sanpham*) và quản lý thao tác *laySanPham()* (giảm/xóa) một cách an toàn

1.2.3. Nhóm lớp hệ thống điều khiển và phụ trợ

Nhóm này chịu trách nhiệm liên kết các nhóm lớp khác, xử lý đầu vào, đầu ra, và quản lý tiến trình trò chơi

- *NguoiChoi:* là lớp trạng thái Game trung tâm, nó liên kết các tài nguyên chính (*Chuong, Vuon, Kho*) và bao bọc các hành động tài chính (*truTien, nhanTien*). Lớp này cũng là điểm gọi logic *NangCap* và *kiemTraTatCaNhiemVu*
- *CuaHang*: triển khai giao diện *MuaBan*. Chịu trách nhiệm thực hiện giao dịch, nơi tiền được trao đổi với sản phẩm/sinh vật, đóng vai trò là cầu nối giữa hệ thống tài chính (*NguoiCho*i) và hệ thống tài nguyên (*Kho*, *Chuong*, *Vuon*).

- *NhiemVu* (abstract) và lớp con: Lớp trừu tượng này cho phép hệ thống linh hoạt kiểm tra điều kiện thông qua *kiemTraHoanThanh* ở lớp *NguoiChoi*. Các lớp con (*NhiemVuNangCap, NhiemVuThuHoach*) mở rộng logic kiểm tra này

1.3. Xác định mối quan hệ và các nguyên lý OOP

Mô hình này áp dụng các nguyên lý lập trình hướng đối tượng (OOP) cơ bản và nâng cao để xây dựng hệ thống

1.3.1. Kế thừa (Inheritance)

Nguyên lý này được sử dụng để xây dựng hệ thống phân cấp *SinhVat* và *Sanpham*, cho phép tái sử dụng mã và định nghĩa hành vi chung tại lớp cha.

Như lớp *Bo.java* kế thừa từ *DongVat.java*, thừa hưởng các thuộc tính *maso, ten, tuoi* và chỉ tập trung định nghĩa hành vi riêng (*keu()* là "Moooo", *an()* là "Ăn Cỏ")

1.3.2. Giao diện và đa hình

Giao diện**:** interface (*Thuhoach, NangCap, MuaBan*) định nghĩa hợp đồng chung cho các hành động cốt lõi.

Đa hình: Cho phép xử lý các đối tượng khác nhau theo cùng một cách thức. Như trong *Chuong.java*, khi gọi thu hoạch, phương thức *chosanpham()* được gọi trên đối tượng *Thuhoach*. Nếu là *Bo* sẽ trả về Sua, nếu là *Ga* sẽ trả về *Trung*

1.3.3. Hợp thành

Mối quan hệ "có một" (has-a) được sử dụng để xây dựng các thực thể lớn từ các thực thể nhỏ hơn và đây là nguyên tắc đóng gói mạnh mẽ

Như *NguoiChoi* có (has-a) *Chuong, Vuon, Kho*. Lớp *NguoiChoi* sẽ là lớp chịu trách nhiệm điều khiển các hoạt động trên các tài nguyên này (ví dụ: *nguoiChoi.nangCapChuong()* sẽ gọi *chuong.nangCap()* sau khi kiểm tra tiền)

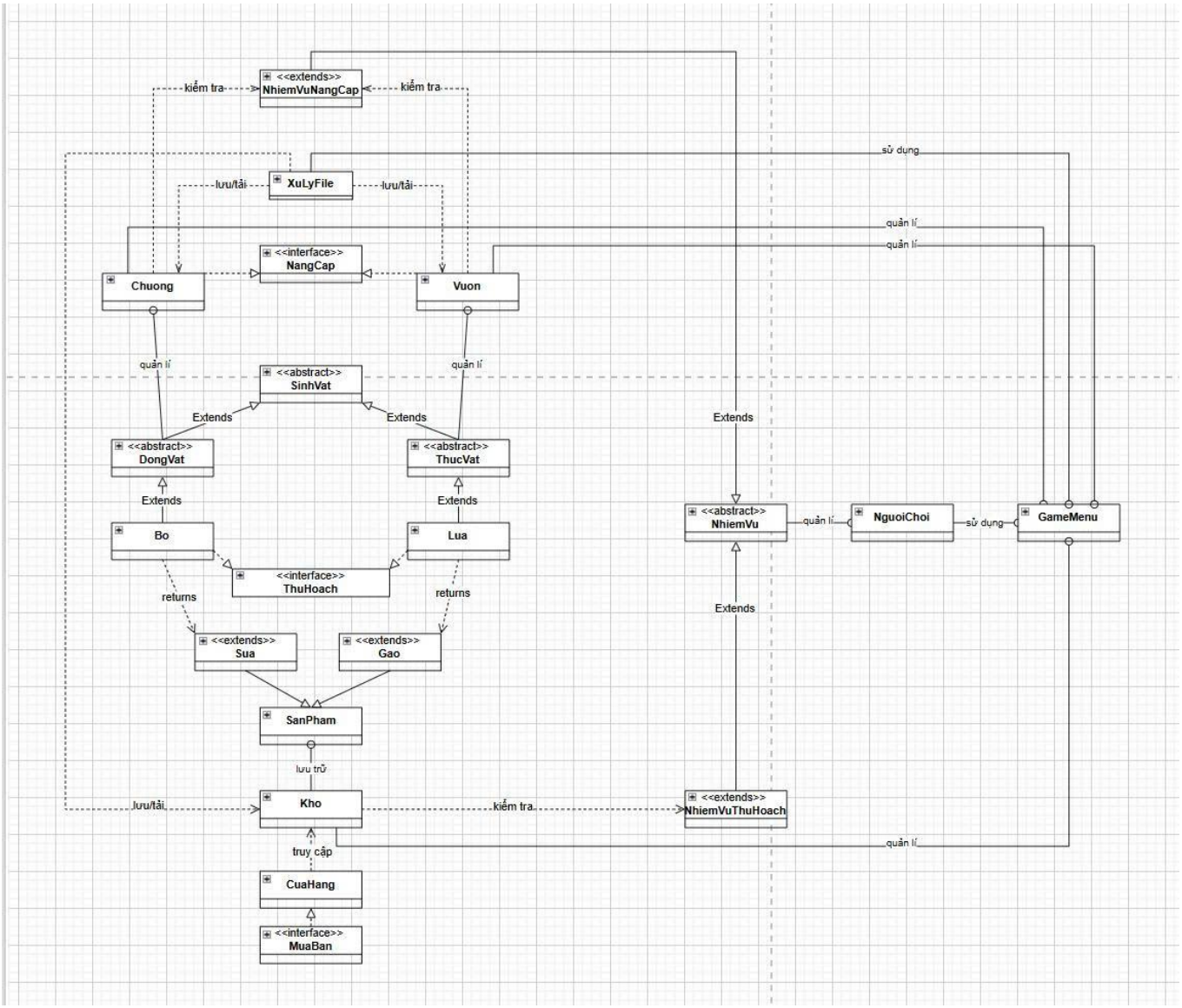
1.3.4. Trừu tượng

Được áp dụng qua các lớp trừu tượng (*SinhVat, DongVat, NhiemVu*) và interface. Cụ thể, lớp *NhiemVu* trừu tượng hóa logic kiểm tra hoàn thành, cho phép người chơi chỉ cần gọi *kiemTraHoanThanh(this)* mà không cần biết đó là nhiệm vụ nâng cấp (*NhiemVuNangCap*)

hay thu hoạch (*NhiemVuThuHoach*). Điều này giúp hệ thống game cốt lõi không bị phụ thuộc vào chi tiết của từng loại nhiệm vụ và dễ dàng mở rộng sau này

CHƯƠNG 2. THIẾT KẾ HỆ THỐNG

2.1. Sơ đồ lớp (UML Class Diagram)



Hình 2.1. Sơ đồ tổng quát UML trò chơi nông trại

Đây là một sơ đồ lớp (Class Diagram) cho trò chơi nông trại, phân tích logic ngắn gọn như sau

- **Hệ thống điều khiển:** lớp GameMenu là trung tâm điều khiển, chịu trách nhiệm quản lý Chuong, Vuon, CuaHang và NguoiChoi
- **Sản xuất:**

- Người chơi quản lý Chuong (nuôi DongVat như Bo) và Vuon (trồng ThucVat như Lua)
- Cả Bo và Lua đều kế thừa từ SinhVat và cùng triển khai (implement) interface ThuHoach

– **Thu hoạch:**

- Khi thu hoạch, Bo trả về Sua, Lua trả về Gao
- Cả Sua và Gao đều là một loại SanPham

– **Lưu trữ và kinh tế:**

- Tất cả SanPham được Kho lưu trữ
- CuaHang truy cập Kho và triển khai interface MuaBan để thực hiện giao dịch

– **Nâng cấp:** Chuong và Vuon có thể được nâng cấp thông qua interface NangCap

– **Hệ thống Nhiệm vụ:**

- Đây là logic kiểm tra (check) trạng thái: NhiemVuNangCap kiểm tra Chuong và Vuon
- NhiemVuThuHoach kiểm tra Kho

– **Lưu trữ:** XuLyFile chịu trách nhiệm lưu và tải dữ liệu của Chuong, Vuon, và Kho

2.2. Thiết kế luồng xử lý (Flow of Control)

Các luồng xử lý sau mô tả hành vi động của hệ thống, đặc biệt nhấn mạnh việc áp dụng nguyên lý đa hình và trừu tượng.

2.2.1. Luồng khởi động và vòng lặp chính

Bước	Mô tả	Lớp tham gia
1	Người dùng chạy chương trình, *GameMenu.main()* được thực thi	*GameMenu.java*
2	Hàm khởi tạo *GameMenu()* được gọi, tạo ra các đối tượng quản lý: *Chuong, Vuon, Kho, CuaHang*, và *NguoiChoi* (với tien khởi tạo là 1000.0). Các đối	*GameMenu.java, NguoiChoi.java, Chuong.java, Vuon.java,*

	tượng này được liên kết với *NguoiChoi* (như *nguoiChoi.setKho(kho))*	*Kho.java*
3	*chayGame()* được gọi, bắt đầu vòng lặp *while(true)*	*GameMenu.java*
4	*hienThiMenu()* in ra các lựa chọn và hệ thống chờ người dùng nhập lệnh qua *Scanner*	*GameMenu.java*
5	Lệnh được xử lý bằng *switch-case* (ví dụ: *case 5*).	*GameMenu.java*
6	Sau khi một hành động (ví dụ thu hoạch) hoàn tất, vòng lặp gọi *nguoiChoi.kiemTraTatCaNhiemVu(),* sau đó quay lại bước 4.	*GameMenu.java, NguoiChoi.java*

Bảng 2.1. Các bước khởi tạo và vòng lặp Game

2.2.2. Luồng thu hoạch (thể hiện đa hình)

- Người chơi chọn "5. Thu hoạch tất cả" tại *GameMenu*
- *GameMenu* gọi *thuHoachTatCa()*
- Phương thức này gọi *chuong.thuHoachTatCa(kho)* và *vuon.thuHoachTatCa(kho)*
- Bên trong *Chuong.java* (tương tự *Vuon.java*)
    - Hệ thống duyệt *List<DongVat>*
    - Với mỗi *dv*, nó kiểm tra *if (dv instanceof Thuhoach)*
    - Nếu đúng, nó gọi phương thức *chosanpham()* từ *interface Thuhoach*
    - Tính đa hình: Nếu *dv* là *Bo, Bo.chosanpham()* được gọi và trả về *new Sua().* Nếu *dv* là *Ga, Ga.chosanpham()* được gọi và trả về *new Trung()*
    - Các sản phẩm này được gộp số lượng và cuối cùng, hệ thống gọi *kho.themSanPham(spGop)* để lưu trữ.

2.2.3. Luồng nâng cấp (thể hiện Interface và điều phối)

- Sau bất kì hành động nào, vòng lặp *chayGame* gọi *nguoiChoi.kiemTraTatCaNhiemVu()*
- Bên trong *NguoiChoi.java*
  - Hệ thống duyệt *List<NhiemVu>*. Với mỗi nv, nó gọi *nv.kiemTraHoanThanh(this*) (truyền chính đối tượng *NguoiChoi* vào)
  - Tính đa hình: Nếu *nv* là *NhiemVuNangCap*, phương thức *NhiemVuNangCap.kiemTraHoanThanh* sẽ chạy. Nó dùng n*guoiChoi* để gọi *nguoiChoi.getChuong().getCapDo()* và so sánh. Còn nếu *nv* là *NhiemVuThuHoach*, phương thức *NhiemVuThuHoach.kiemTraHoanThanh* sẽ chạy. Nó dùng *nguoiChoi* để gọi *nguoiChoi.getKho().demSoLuong()* và so sánh
- Nếu điều kiện đúng, nhiệm vụ đó sẽ tự đặt *this.hoanThanh = true* và *NguoiChoi* được cộng *tienThuong*

CHƯƠNG 3. CÀI ĐẶT VÀ MÔ TẢ CÁC CHỨC NĂNG CHÍNH

3.1. Triển khai cài đặt và bộ khung

3.1.1. Giới thiệu cấu trúc dự án và luồng điều khiển

Dự án game nông trại được xây dựng trên một kiến trúc phần mềm vững chắc, áp dụng sâu sắc các nguyên tắc của lập trình hướng đối tượng (OOP) để tạo nên một hệ thống vừa linh hoạt vừa có trật tự. Trung tâm của toàn bộ ứng dụng là lớp *GameMenu*, đóng vai trò là "xương sống" và bộ điều khiển chính. Lớp này chịu trách nhiệm xử lý vòng lặp chính của trò chơi (game loop), tiếp nhận thông tin đầu vào từ người dùng và điều phối các hành động. Liên kết chặt chẽ với *GameMenu* là đối tượng *NguoiChoi*, một lớp đại diện cho người dùng, lưu trữ các thông tin cốt lõi như tên và lượng tiền. Quan trọng hơn, *NguoiChoi* đóng vai trò là chủ sở hữu của các tài nguyên chính trong game, bao gồm một *Chuong*, một *Vuon*, và một *Kho*, đồng thời cũng là người quản lý danh sách các *NhiemVu* cần thực hiện.

Hệ thống quản lý tài nguyên được thiết kế rõ ràng thông qua ba lớp quản lý chính. Lớp *Chuong* chịu trách nhiệm lưu trữ và xử lý logic cho một danh sách các đối tượng *DongVat*. Tương tự, lớp *Vuon* quản lý một danh sách các đối tượng *ThucVat*. Cả hai lớp này thể hiện một quy tắc chung của trò chơi thông qua việc thực thi (implement) *interface NangCap*, cho phép chúng được nâng cấp để mở rộng sức chứa hoặc tính năng. Cuối cùng, lớp *Kho* là nơi lưu trữ các *SanPham* đã được thu hoạch, hoạt động như một kho chứa chung cho tất cả thành phẩm mà người chơi kiếm được. Cấu trúc này, nơi các lớp quản lý danh sách (*Chuong, Vuon, Kho*) được tách biệt, giúp cho việc thêm, xóa, và tìm kiếm đối tượng trở nên gọn gàng và dễ bảo trì

Phần cốt lõi của dữ liệu xoay quanh hai cây kế thừa chính là *SinhVat* và *SanPham*, định nghĩa các đối tượng dữ liệu cốt lõi, tận dụng tính kế thừa và đa hình để dễ dàng quản lý các loại vật nuôi, cây trồng và sản phẩm khác nhau.

GameMenu và *NguoiChoi* tạo thành bộ điều khiển và mô hình trung tâm. *NguoiChoi* kết nối với các tài nguyên thông qua các lớp quản lý dữ liệu (Data Managers) là *Chuong* (quản lý vật nuôi), *Vuon* (quản lý cây trồng) và *Kho* (quản lý sản phẩm).

Cuối cùng, các quy tắc và tính năng của trò chơi được định nghĩa thông qua các interface chuyên biệt như *NangCap* và *MuaBan*, cùng với các lớp dịch vụ như *CuaHang* và *NhiemVu*. Những thành phần này định nghĩa các cơ chế tương tác, kết nối tất cả các thành phần lại với nhau thành một trò chơi hoàn chỉnh và có thể mở rộng

Luồng điều khiển: các quy tắc và tính năng tương tác được định nghĩa qua các lớp và interface chuyên biệt

- Khởi tạo và quản lý chung:
  - Khởi động: *GameMenu* là lớp trung tâm, bắt đầu chương trình.
  - Nạp dữ liệu: *GameMenu* gọi *XuLyFile* để tải dữ liệu cũ (nếu có) về trạng thái của *Chuong, Vuon, Kho*, và thông tin *NguoiChoi*
  - Điều hướng: *GameMenu* hiển thị các tùy chọn và điều phối tương tác của người chơi đến các phân hệ khác (*NhiemVu, CuaHang, Chuong*)
- Luồng sản xuất và thu hoạch : quá trình tạo ra sản phẩm nông nghiệp được thực hiện thông qua tương tác của *NguoiCho*i với các công trình quản lý.
  - Nuôi trồng: *NguoiChoi* tác động vào *Chuong* (chứa *DongVat* như *Bo*) hoặc *Vuon* (chứa *ThucVa*t như *Lua*)
  - Thu hoạch: Khi các *SinhVat* đủ điều kiện, chúng thực thi phương thức từ interface ThuHoach, hệ thống trả về đối tượng *SanPham* cụ thể (như *Bo* trả về *Sua*, *Lua* trả về *Gao*)
  - Lưu trữ: Các *SanPham* sau khi thu hoạch sẽ được chuyển vào và lưu trữ trong *Kho*
- Luồng Nhiệm vụ : hệ thống nhiệm vụ hoạt động theo cơ chế "kiểm tra" (*Check*) trạng thái của các đối tượng khác
  - Nhiệm vụ thu hoạch (NhiemVuThuHoach): liên tục hoặc định kỳ kiểm tra Kho để xem số lượng *SanPham* đã đủ chỉ tiêu yêu cầu chưa.
  - Nhiệm vụ nâng cấp (*NhiemVuNangCap*): kiểm tra cấp độ hiện tại của *Chuong* hoặc *Vuon* để xác nhận hoàn thành nhiệm vụ mở rộng quy mô.
- Luồng Lưu trữ: luồng này đảm bảo tiến trình của người chơi được duy trì qua các lần chơi
  - Ghi file: khi người chơi chọn Lưu hoặc Thoát từ *GameMenu*, lớp *XuLyFile* sẽ thu thập dữ liệu hiện tại từ *Chuong, Vuon, Kho* và ghi xuống bộ nhớ

- Tái tạo: lớp *LuuQuaTrinh* chịu trách nhiệm tái tạo các đối tượng và khôi phục trạng thái trò chơi từ dữ liệu đã lưu khi khởi động.

3.1.2. Mô tả triển khai lớp

Cài đặt các lớp quản lý danh sách (CRUD): Các công trình chính trong nông trại (*Chuong và Vuon*) đóng vai trò là các đối tượng quản lý danh sách. Việc cài đặt các phương thức cơ bản (CRUD) đảm bảo người chơi có thể tương tác và quản lý các sinh vật một cách hiệu quả

- Quản lý Sinh vật (động vật và thực vật)
  - Các lớp *Chuong* và *Vuon* quản lý danh sách sinh vật bằng cách sử dụng cấu trúc *List<DongVat>* và *List<Thucvat>* tương ứng
  - Các phương thức CRUD cơ bản (*themDongVat()*/ *themThucVat())* ,được cài đặt để quản lý các danh sách này. Đặc biệt, phương thức *themDongVat()* trong lớp *Chuong* và *themThucVat()* trong lớp *Vuon* có tích hợp logic kiểm tra *sucChuaToiDa()* (từ interface *NangCap*) để đảm bảo giới hạn sức chứa của công trình
- Quản lý Kho (logic xếp chồng - Stacking)
  - Lớp *Kho* quản lý hàng tồn kho bằng *List<Sanpham>,* tập trung vào việc tối ưu hóa bộ nhớ thông qua logic xếp chồng (stacking)
  - Phương thức *themSanPham(Sanpham sp)* triển khai bằng cách sử dụng *timSanPham*(có tích hợp xử lý dấu tiếng Việt) để kiểm tra sự tồn tại của sản phẩm. Nếu sản phẩm đã tồn tại thì chỉ *tangsoluong()* của đối tượng *Sanpham* hiện có lên. Nếu như sản phẩm đó chưa tồn tại thì thêm đối tượng *Sanpham* mới vào danh sách
  - Nguyên tắc này đảm bảo tính đóng gói mạnh mẽ và tối ưu hóa bộ nhớ bằng cách tránh tạo các đối tượng sản phẩm trùng lặp

Cài đặt lưu và tải Game với lớp *XuLyFile.java* đóng vai trò là bộ phận xử lý đầu ra, đầu vào (I/O) chính, chịu trách nhiệm thực hiện chức năng lưu (*luuGame*) và tải (*taiGame*) trò chơi.

- Tính năng *luuGame*
  - Sử dụng lớp *PrintWriter* (Java I/O) để ghi dữ liệu tuần tự vào file *game_save.txt*

- Các thuộc tính cơ sở (*tien, capDo*) được ghi trực tiếp
- Logic lưu trữ sử dụng vòng lặp để duyệt danh sách của *Chuong, Vuon, Kho* và ghi dữ liệu chi tiết của từng đối tượng ra file theo định dạng key:value (DONGVAT:SV001,Bo,2)

– Tính năng *taiGame*
  - Sử dụng *BufferedReader* (đọc văn bản từ một luồng đầu vào) để đọc file *game_save.txt*, phân tích cú pháp từng dòng và nạp dữ liệu thô vào đối tượng trung gian *GameData*
  - *LuuQuaTrinh* (sử dụng *GameData*) chịu trách nhiệm tái tạo đối tượng: dựa vào tên lớp được lưu, nó gọi constructor cụ thể (*new Bo(ten, tuoi)* hay *new Lua(ten, tuoi)*) để khôi phục trạng thái đối tượng, đảm bảo tính toàn vẹn của kế thừa và đa hình sau khi tải

3.2. Mô tả chức năng và chứng minh nguyên lý OOP

Phần này phân tích chuyên sâu việc áp dụng các nguyên lý lập trình hướng đối tượng (OOP) như kế thừa, đa hình, và trừu tượng hóa thông qua hai mô hình chức năng quan trọng định hình cấu trúc và tính mở rộng của trò chơi: mô hình sinh vật và mô hình nhiệm vụ.

3.2.1. Cài đặt kế thừa và đa hình (mô hình sinh vật)

Tính kế thừa và trừu tượng

– Lớp *SinhVat.java* (abstract)
  - Khai báo *protected (maso, ten, tuoi)*: mục đích là cho phép các lớp con (*DongVat, Thucvat*) có thể truy cập trực tiếp các thuộc tính này để thao tác, trong khi vẫn ẩn chúng khỏi các lớp không liên quan, thực thi nguyên lý đóng gói
  - Khai báo *public abstract void phattrien():* mục đích là bắt buộc mọi lớp concrete (lớp không abstract) kế thừa từ *SinhVa*t phải định nghĩa logic phát triển riêng của mình, đảm bảo tính đầy đủ của mô hình sinh học

– Mô hình kế thừa lớp trung gian
  - *DongVat* (kế thừa *SinhVat*): Bổ sung thuộc tính *protected String Loaithucan* (loại thức ăn) nhằm chuyên biệt hóa dữ liệu cần thiết cho nhóm động vật.

Việc khai báo protected giúp các lớp con như *Bo, Ga,…* có thể truy cập *Loaithucan* trong phương thức a*n()* (ăn) của mình

- *Thucvat* (kế thừa *SinhVat*): bổ sung thuộc tính *protected int giaiDoanPhatTrien* và *protected boolean daTuoiNuoc* nhằm mô hình hóa chu kỳ sống và trạng thái cần thiết của cây trồng. Việc này là cần thiết để kiểm soát hành vi *tuoiNuoc()* và *thuHoach*

– Chuỗi kế thừa hoàn chỉnh: Chuỗi kế thừa được thiết lập: *SinhVat* → *DongVat* (abstract) → *Bo, Ga, Heo*. Tương tự: *SinhVat* → *Thucvat* (abstract) → *Lua, Ngo, Mamxoi*. Việc sử dụng từ khóa *extends* cho phép các lớp con tái sử dụng toàn bộ cấu trúc dữ liệu và logic không trừu tượng từ lớp cha

Tính đa hình

– Trừu tượng hóa hành vi (trong *DongVat.java* và Thucvat.java)
  - *DongVat.java* khai báo phương thức trừu tượng *public abstract void keu()* và *an()*
  - Thucvat.java khai báo *public abstract void tuoiNuoc()*
  - Mục đích là định nghĩa hành vi chung (kêu, ăn, tưới nước), nhưng giao việc tùy biến cụ thể (kêu "*Mooo*", Lúa có thông báo tưới nước riêng) cho các lớp con
– Ghi đè hành vi (trong *Bo.java, Ga.java,…*)
  - Các lớp con sử dụng annotation *@Override* để cung cấp thân (body) cho các phương thức trừu tượng, hoàn tất quá trình định nghĩa hành vi riêng
– Ngữ cảnh thực thi và Liên kết Động (trong Chuong.java và Vuon.java)
  - Lớp *Chuong.java* quản lý *List<DongVat>*. Khi gọi *dv.keu()* hoặc *dv.an()* trên từng phần tử, cơ chế Liên kết Động (Dynamic Binding) của Java sẽ tự động xác định đối tượng thực sự (*Bo* hay *Ga*) để gọi đúng hàm đã được *@Override*, chứng minh nguyên lý đa hình
– Đa hình Interface
  - Interface *Thuhoach.java* định nghĩa hợp đồng *Sanpham chosanpham()*
  - Cả *DongVat* (*Bo, Ga, Heo*) và Thucvat (*Lua, Ngo, Mamxoi*) đều implements interface này. Điều này cho phép phương thức *thuHoachTatCa()* (trong

Chuong và *Vuon*) xử lý các loại sinh vật khác nhau theo cùng một giao diện để nhận về sản phẩm, đảm bảo tính linh hoạt của mô hình sản xuất.

3.2.2. Cài đặt hệ thống nhiệm vụ và logic điều phối

- Lớp *NhiemVu* (abstract): đây là lớp cha, định nghĩa khung sườn cho các nhiệm vụ.

Thành phần	Tên	Kiểu dữ liệu	Mô tả
Thuộc tính	*moTa*	*String*	Mô tả nội dung nhiệm vụ (được bảo vệ/ *protected*)
(protected)	*hoanThanh*	*boolean*	Trạng thái của nhiệm vụ (*True*: Đã xong, *False*: Chưa)
	tienThuong	*double*	Số tiền thưởng nhận được khi hoàn thành
Phương thức	*NhiemVu(...)*	*Constructor*	Khởi tạo nhiệm vụ với mô tả và tiền thưởng
(public)	*kiemTraHoanThanh()*	*void*	Phương thức trừu tượng (abstract), lớp con phải tự định nghĩa logic
	getMoTa()	*String*	Lấy nội dung mô tả nhiệm vụ
	isHoanThanh()	*boolean*	Kiểm tra trạng thái hoàn thành hiện tại
	setHoanThanh()	*void*	Cập nhật trạng thái hoàn thành

Bảng 3.1. Bảng lớp trừu tượng nhiệm vụ

- Lớp NhiemVuThuHoach (kế thừa NhiemVu): lớp này chuyên xử lý các nhiệm vụ liên quan đến việc thu hoạch nông sản trong kho.

Thành phần	Tên	Kiểu dữ liệu	Mô tả
Thuộc tính	t*enSanPhamYeuCau*	String	Tên loại sản phẩm cần thu hoạch (Ví dụ: "*Mamxo*i")
(private)	*soLuongYeuCau*	int	Số lượng sản phẩm cần có trong kho để hoàn thành
Phương thức	*NhiemVuThuHoach(...)*	Constructor	Khởi tạo nhiệm vụ thu hoạch, gọi *super()* từ lớp cha
(public)	*kiemTraHoanThanh()*	void	Ghi đè (*Override*): Kiểm tra Kho của người chơi xem đủ số lượng sản phẩm yêu cầu chưa

Bảng 3.2. Bảng lớp nhiệm vụ thu hoạch

– *Lớp NhiemVuNangCap* (kế thừa NhiemVu): lớp này chuyên xử lý các nhiệm vụ liên quan đến cấp độ của công trình (Vườn/Chuồng).

Thành phần	Tên	Kiểu dữ liệu	Mô tả
Thuộc tính	*capDoYeuCau*	int	Cấp độ mục tiêu cần đạt được (Ví dụ: Level 2)
(private)	*loaiCongTrinh*	String	Loại công trình cần kiểm tra ("*VUON*" hoặc "*CHUONG*")
Phương thức	*NhiemVuNangCap(...)*	Constructor	Khởi tạo nhiệm vụ nâng cấp, gọi *super()* từ lớp cha
(public)	*kiemTraHoanThanh()*	void	Ghi đè (*Override*): Truy cập Vườn/Chuồng của người chơi để kiểm tra cấp độ hiện tại

Bảng 3.3. Bảng lớp nhiệm vụ nâng cấp

– Lớp *CuaHang* (thực thi giao diện *MuaBan*): lớp này quản lý logic giao dịch mua vật phẩm và bán nông sản.

Thành phần	Tên	Kiểu dữ liệu	Mô tả

Phương thức	*CuaHang()*	*Constructor*	Khởi tạo đối tượng Cửa Hàng
(public)	*banSanPham()*	*void*	Ghi đè: Kiểm tra kho người chơi, trừ sản phẩm và cộng tiền (Bán cho cửa hàng)
	muaVatPham()	*void*	Ghi đè: Kiểm tra tiền người chơi, trừ tiền và thêm vật phẩm vào Vườn/Chuồng (Mua từ cửa hàng)

Bảng 3.4. Bảng lớp cửa hàng

– *Lớp XuLyFile* (Xử lý lưu trữ): đây là lớp chịu trách nhiệm quản lý việc đọc/ghi dữ liệu của hệ thống ra file vật lý đảm bảo tính bền vững của dữ liệu.

Thành phần	Tên	Kiểu dữ liệu	Mô tả
Thuộc tính	*FILE_SAVE*	*String (static final)*	Tên file cố định dùng để lưu dữ liệu game ("*game_save.txt*").
Phương thức	*luuGame(...)*	*void*	Ghi toàn bộ trạng thái hiện tại của *Chuong, Vuon, Kho* và tiền người chơi xuống file văn bản.
	taiGame()	*GameData*	Đọc từng dòng từ file save, phân tích cú pháp (parse) và nạp dữ liệu vào đối tượng *GameData* để khôi phục game.

Bảng 3.5. Bảng lớp xử lý file

– Lớp *Kho* (quản lý tài nguyên): Lớp trung tâm lưu trữ tất cả các đối tượng Sanpham. Đóng vai trò là "bộ đệm" trung gian giữa sản xuất và thương mại.

Thành phần	Tên	Kiểu dữ liệu	Mô tả
Thuộc tính	*danhSachSanPham*	*List<Sanpham>*	Danh sách chứa các đối tượng sản phẩm hiện có trong kho.

	maKho	*String*	Mã định danh duy nhất của kho (ví dụ: KHO1), được tạo tự động.
Phương thức	*themSanPham(Sanpham)*	*void*	Thêm sản phẩm vào kho. Nếu sản phẩm đã tồn tại, tự động cộng dồn số lượng thay vì tạo mới.
	laySanPham(...)	*boolean*	Kiểm tra số lượng tồn và trừ bớt sản phẩm khi bán hoặc sử dụng. Trả về false nếu không đủ hàng.
	timSanPham(String)	*Sanpham*	Tìm và trả về đối tượng sản phẩm dựa trên tên gọi.
	hienThiTonKho()	*void*	In ra danh sách các sản phẩm, số lượng và tổng giá trị tài sản hiện có trong kho.
	coDuSanPham(...)	*boolean*	Phương thức kiểm tra (Validation) xem kho có đủ số lượng mặt hàng yêu cầu không.

Bảng 3.6. Bảng lớp kho

- Lớp *Vuon* (Quản lý trồng trọt): lớp quản lý các đối tượng *Thucvat*. Lớp này hiện thực hóa interface *NangCap* để mở rộng quy mô sản xuất

Thành phần	Tên	Kiểu dữ liệu	Mô tả
Thuộc tính	*danhSachCayTrong*	*List<Thucvat>*	Danh sách lưu trữ các cây trồng hiện có trong vườn.
	capDo	*int*	Cấp độ hiện tại của vườn (ảnh hưởng đến sức chứa).
	sucChuaToiDa	*int*	Số lượng cây trồng tối đa mà vườn có thể chứa tại cấp độ hiện tại.

	chiPhiNangCap	*double*	Số tiền cần thiết để nâng cấp vườn lên cấp tiếp theo.
Phương thức	*themThucVat(...)*	*boolean*	Thêm cây mới vào vườn sau khi kiểm tra sức chứa tối đa.
	tuoiNuocTatCa()	*void*	Duyệt danh sách và kích hoạt hành động tưới nước cho toàn bộ cây trồng.
	thuHoachTatCa(Kho)	*void*	Thu hoạch toàn bộ cây, sử dụng Map để gộp các sản phẩm trùng nhau trước khi đẩy vào Kho.
	nangCap()	*void*	Tăng cấp độ vườn, mở rộng sức chứa (+5 slot) và tăng chi phí cho lần nâng cấp sau.
	syncCapDo(...)	*void*	Đồng bộ lại cấp độ và tính toán lại sức chứa khi nạp game từ file save.

Bảng 3.7. Bảng lớp vườn

- Lớp *Chuong* (Quản lý chăn nuôi): lớp quản lý các đối tượng *DongVat.* Tương tự như *Vuon*, lớp này cũng hiện thực hóa interface *NangCap* nhưng có logic xử lý vật nuôi khác biệt.

Thành phần	Tên	Kiểu dữ liệu	Mô tả
Thuộc tính	*danhSachDongVat*	*List<DongVat>*	Danh sách lưu trữ các con vật đang nuôi trong chuồng.
	trangThai	*String*	Mô tả tình trạng chung của chuồng (ví dụ: "Bình thường").
	sucChuaToiDa	*int*	Giới hạn số lượng vật nuôi (Cấp 1 chứa tối đa 5 con).
Phương thức	*themDongVat(...)*	*boolean*	Thêm vật nuôi mới vào

			chuồng nếu chưa vượt quá sức chứa.
	choTatCaAn()	*void*	Duyệt danh sách và kích hoạt hành động ăn cho toàn bộ vật nuôi.
	thuHoachTatCa(Kho)	*void*	Thu hoạch sản phẩm (Sữa/Trứng...). Xử lý đặc biệt: Xóa vật nuôi khỏi danh sách nếu là loại thu hoạch một lần (ví dụ: Heo).
	nangCap()	*void*	Tăng cấp độ chuồng, mở rộng sức chứa (+3 slot) và cập nhật chi phí nâng cấp theo hệ số nhân 1.5.

Bảng 3.8. Bảng lớp chuồng

CHƯƠNG 4. KẾT QUẢ VÀ GIAO DIỆN

4.1. Mô tả giao diện

Giao diện người dùng của dự án là giao diện dòng Lệnh (Console/Terminal). Thay vì sử dụng đồ họa, hệ thống sử dụng văn bản và menu số hóa, được điều khiển bởi lớp *GameMenu* (bộ điều khiển trung tâm), để quản lý toàn bộ trải nghiệm người chơi

4.1.1. Vai trò chiến lược của lớp GameMenu

Lớp *GameMenu* hoạt động như Controller (bộ điều khiển) trong mô hình kiến trúc ứng dụng, thực hiện vai trò điều phối trung tâm

- Khởi tạo và liên kết Game: ngay khi trò chơi bắt đầu, *GameMenu* chịu trách nhiệm tạo ra tất cả các đối tượng cốt lõi: *NguoiChoi* (người chơi), *Chuong* (chuồng), *Vuon* (vườn), *Kho* (kho), *CuaHang* (cửa hàng). Quan trọng hơn, nó thực hiện liên kết dữ liệu bằng việc gắn các công trình (*Chuong, Vuon, Kho*) vào đối tượng *NguoiChoi*. Điều này cho phép *NguoiChoi* trở thành trung tâm quản lý tài sản
- Vòng lặp và điều khiển lệnh: *GameMenu* duy trì một vòng lặp liên tục (*while (dangChoi)*) để hiển thị menu và chờ lệnh từ người dùng. Hệ thống sử dụng cấu trúc *switch* để điều hướng, nhận số người dùng nhập và gọi đúng hàm xử lý chức năng tương ứng (ví dụ: gõ 1 gọi *themDongVatMenu()*)

```
--- GAME NÔNG TRẠI ---
Tiền: 1000.0
1. Thêm động vật (Trồng con giống)
2. Trồng cây (Trồng hạt giống)
3. Vào cửa hàng (Mua/Bán)
4. Lưu game
5. Thu hoạch tất cả
6. Hành động (Cho ăn, Tưới nước)
7. Xem tổng quan trang trại
8. Nâng cấp công trình (Chuồng/Vườn)
9. Thoát
Chọn: 1

--- THÊM ĐỘNG VẬT ---
1. Thêm Gà
2. Thêm Bò
3. Thêm Heo
Chọn loại:
```

Hình 4.1. Cơ chế điều khiển lệnh (Switch-Case) điều hướng từ Menu Chính

– Xử lý nhập liệu an toàn: giao diện được trang bị cơ chế *try-catch* (kiểm soát lỗi) khi đọc lệnh. Nghĩa là nếu như người dùng vô tình nhập kí tự chữ (a,b,…) thay vì số thì game sẽ không bị dừng đột ngột mà chỉ thông báo lỗi sau đó tiếp tục vòng lặp, đảm bảo tính ổn định của ứng dụng

```
--- GAME NÔNG TRẠI ---
Tiền: 1000.0
1. Thêm động vật (Trồng con giống)
2. Trồng cây (Trồng hạt giống)
3. Vào cửa hàng (Mua/Bán)
4. Lưu game
5. Thu hoạch tất cả
6. Hành động (Cho ăn, Tưới nước)
7. Xem tổng quan trang trại
8. Nâng cấp công trình (Chuồng/Vườn)
9. Thoát
Chọn: a
[X] Lựa chọn không hợp lệ (Phải là số)!
```

Hình 4.2. Giao diện Menu chính và thông báo lỗi xử lý đầu vào

4.1.2. Phân tích các Chức năng giao tiếp chuyên biệt

Các phương thức riêng tư (*private methods*) trong lớp GameMenu chịu trách nhiệm giao tiếp với người dùng và, tuân theo nguyên tắc ủy thác, chuyển yêu cầu tới các lớp chuyên môn để xử lý logic game (như tính toán chi phí,...) nhằm đảm bảo giao diện được phân cấp rõ ràng

Chức năng người dùng	**Lớp/ phương thức được ủy thác**	**Tính chất kỹ thuật**
Thêm động vật/ trồng cây (lệnh 1, 2)	*chuong.themDongVat(),vuon.themThucVat()*	Tạo đối tượng Cụ thể (*new Ga()*, *new Lua()*) và gọi chuồng/ vườn để kiểm tra sức chứa và thêm vào danh sách
Vào cửa hàng (lệnh 3)	*cuaHang.muaVatPham(), banSanPham()*	Giao diện thu thập tên/ số lượng, sau đó gọi *CuaHang* và truyền vào đối tượng *NguoiChoi* để lớp *CuaHang* tự xử lý logic trừ tiền và kiểm tra kho
Lưu Game (lệnh 4)	*xuLyFile.luuGame(...)*	Lấy toàn bộ trạng thái (chuồng, vườn, kho, tiền) và ủy thác cho module *XuLyFile* để ghi dữ liệu vào ổ đĩa
Thu hoạch/ hành động (lệnh 5, 6)	*vuon.thuHoachTatCa(kho), chuong.choTatCaAn()*	*GameMenu* chỉ ra lệnh "hành động đồng loạt". Nó không cần biết cách thức thu hoạch/ cho ăn diễn ra thế nào (nguyên tắc đa hình)

Xem tổng quan trang trại (lệnh 7)	*nguoiChoi.hienThiTrangThai()*	Lớp *NguoiChoi* tự tổng hợp toàn bộ thông tin tài chính, nhiệm vụ, và trạng thái công trình để in ra màn hình
Nâng cấp công trình (lệnh 8)	*nguoiChoi.nangCapChuong()*	Giao diện hiển thị chi phí động bằng cách gọi hàm *getChiPhiNangCap() từ Chuong/Vuon* và ủy thác cho lớp *NguoiChoi* lo việc kiểm tra tiền, trừ tiền, và thực hiện nâng cấp công trình.

Bảng 4.1. Minh họa nguyên tắc ủy thác nhiệm vụ từ lớp gamemenu sang các lớp chuyên môn

4.2. Các kết quả đạt được

4.2.1. Trạng thái khởi tạo và điều hướng cơ bản

Hệ thống bắt đầu bằng việc hiển thị luật chơi và sau đó là Menu chính. Lớp *GameMenu* chịu trách nhiệm xử lý các lỗi nhập liệu ngay tại tầng giao diện.

- Trạng thái khởi tạo: Khi chạy *GameMenu.java*, Console sẽ hiển thị Luật chơi (từ *hienThiLuatChoi()*) và ngay sau đó, Menu chính đầu tiên xuất

hiện

```
[+] Người chơi Nông dân zui zẻ đã bắt đầu!

==================================
=== CHÀO MỪNG ĐẾN NÔNG TRẠI ===
==================================
Mục tiêu của bạn là quản lý nông trại, kiếm tiền và hoàn thành nhiệm vụ.
Bạn bắt đầu với 1000.0 Tiền.

--- CÁCH CHƠI CƠ BẢN ---
1. KIẾM TIỀN:
   - Dùng mục [1] hoặc [2] để thêm (miễn phí) động vật/cây trồng.
   - Dùng mục [5] (Thu hoạch) để lấy sản phẩm (Sữa, Gạo...) vào Kho.
   - Dùng mục [3] (Cửa hàng) -> Bán, để bán sản phẩm từ Kho lấy tiền.
2. NÂNG CẤP:
   - Nông trại có giới hạn sức chứa (Chuồng: 5, Vườn: 10).
   - Dùng mục [8] (Nâng cấp) để tiêu tiền, tăng sức chứa.
3. NHIỆM VỤ:
   - Dùng mục [7] (Xem tổng quan) để theo dõi nhiệm vụ.
   - Nhiệm vụ (ví dụ: 'Thu hoạch 5 Sữa') sẽ tự động hoàn thành khi bạn đủ điều kiện.

Chúc bạn chơi vui!
```

Hình 4.3. Trạng thái khởi tạo game và hiển thị luật chơi cơ bản

– Xử lý lỗi nhập liệu

Mục tiêu	**Các bước thực hiện (Input & Output)**	**Kết quả cuối cùng**
Lỗi (Nhập chữ)	1. **Hệ thống:** Chọn: 2. **Nhập:** abc 3. **Hệ thống:** [X] Lựa chọn không hợp lệ (Phải là số)!	Menu chính được hiển thị lại.
Lỗi (Số ngoài dải)	1. **Hệ thống:** Chọn: 2. **Nhập:** 10 3. **Hệ thống:** Lựa chọn không hợp lệ!!!!	Menu chính được hiển thị lại.
Thoát (Lựa chọn 9)	1. **Hệ thống:** Chọn: 2. **Nhập:** 9 3. **Hệ thống:** Đang thoát...	Chương trình kết thúc.

Bảng 4.2. Các trường hợp kiểm thử xử lý đầu vào và thoát chương trình tại menu chính

4.2.2. Quản lý sinh vật và tồn kho

– Thêm động vật (lựa chọn 1)

Mục tiêu	Các bước thực hiện (Input & Output)	Kết quả cuối cùng
Thêm Gà (Happy Path)	1. **Nhập:** 1 2. **Hệ thống:** Chọn loại: 3. **Nhập:** 1 (Gà) 4. **Hệ thống:** Nhập tên: 5. **Nhập:** GaCon 6. **Hệ thống:** Nhập tuổi: 7. **Nhập:** 1 8. **Hệ thống:** Trạng thái Chuồng: [1/5]	Quay lại menu chính. Chuồng có 1 động vật.
Lỗi (Menu con nhập chữ)	1. **Nhập:** 1 2. **Hệ thống:** Chọn loại: 3. **Nhập:** abc 4. **Hệ thống:** [X] Lựa chọn không hợp lệ!	Quay lại menu chính. Không thêm gì cả.
Lỗi (Quá sức chứa)	(Giả định chuồng đã [5/5]) 1. **Nhập:** 1 2. **Hệ thống:** Chọn loại: 3. **Nhập:** 1 4. ... (Nhập tên, tuổi) 5. **Hệ thống:** (Không in ra dòng trạng thái)	Quay lại menu chính. Trạng thái chuồng vẫn [5/5].

Bảng 4.3. Các trường hợp kiểm thử chức năng "thêm động vật" (lệnh 1) bao gồm xử lý thành công và lỗi

– Trồng cây (lựa chọn 2)

Mục tiêu	Các bước thực hiện (Input & Output)	Kết quả cuối cùng

Trồng Lúa (Happy Path)	1. **Nhập:** 2 2. **Hệ thống:** Chọn loại: 3. **Nhập:** 1 (Lúa) 4. **Hệ thống:** Nhập tên cây: 5. **Nhập:** LuaMoi 6. **Hệ thống:** Trạng thái Vườn: [1/10]	Quay lại menu chính. Vườn có 1 cây.
Lỗi (Menu con nhập chữ)	1. **Nhập:** 2 2. **Hệ thống:** Chọn loại: 3. **Nhập:** xyz 4. **Hệ thống:** [X] Lựa chọn không hợp lệ!	Quay lại menu chính. Không trồng gì cả.

Bảng 4.4. Các trường hợp kiểm thử chức năng "trồng cây" (lệnh 2) bao gồm xử lý thành công và lỗi đầu vào.

– Cửa hàng (lựa chọn 3)

Mục tiêu	**Các bước thực hiện (Input & Output)**	**Kết quả cuối cùng**
Mua (Happy Path)	1. **Nhập:** 3 2. **Hệ thống:** Chọn: 3. **Nhập:** 1 (Mua) 4. **Hệ thống:** Nhập tên vật phẩm... 5. **Nhập:** BoCon 6. **Hệ thống:** Nhập số lượng: 7. **Nhập:** 1 8. **Hệ thống:** (Thông báo mua thành công)	Quay lại menu chính. Tiền bị trừ (ví dụ: Tiền: 800.0).

Bán (Kho trống)	1. **Nhập:** 3 2. **Hệ thống:** Chọn: 3. **Nhập:** 2 (Bán) 4. **Hệ thống:** Nhập tên sản phẩm... 5. **Nhập:** Sữa 6. **Hệ thống:** Nhập số lượng: 7. **Nhập:** 1 8. **Hệ thống:** (Thông báo lỗi kho trống)	Quay lại menu chính. Tiền không đổi (1000.0).
Bán (Happy Path)	(Giả định đã có 'Sữa' trong kho) 1. **Nhập:** 3 2. **Hệ thống:** Chọn: 3. **Nhập:** 2 (Bán) 4. **Hệ thống:** Nhập tên sản phẩm... 5. **Nhập:** Sữa 6. **Hệ thống:** Nhập số lượng: 7. **Nhập:** 1 8. **Hệ thống:** (Thông báo bán thành công)	Quay lại menu chính. Tiền tăng lên (ví dụ: Tiền: 1050.0).
Lỗi (Nhập chữ ở số lượng)	1. **Nhập:** 3 2. **Hệ thống:** Chọn: 3. **Nhập:** 1 (Mua) 4. ... (Nhập tên) 5. **Hệ thống:** Nhập số lượng: 6. **Nhập:** abc 7. **Hệ thống:** [X] Số lượng không hợp lệ!	Quay lại menu chính. Giao dịch bị hủy.

Bảng 4.5. Các trường hợp kiểm thử chức năng "vào cửa hàng" (lệnh 3) bao gồm mua, bán và xử lý lỗi

4.2.3. Tác vụ và nâng cấp công trình

- Tác vụ (lựa chọn 4, 5, 6)

Mục tiêu	Các bước thực hiện (Input & Output)	Kết quả cuối cùng
Lưu Game (Lựa chọn 4)	1. **Nhập:** 4 2. **Hệ thống:** (Thông báo từ xuLyFile.luuGame)	Quay lại menu chính.
Thu Hoạch (Lựa chọn 5)	1. **Nhập:** 5 2. **Hệ thống:** --- Bắt đầu thu hoạch --- 3. **Hệ thống:** Đang gọi thu hoạch Vườn... 4. **Hệ thống:** Đang gọi thu hoạch Chuồng...	Quay lại menu chính. Sản phẩm được thêm vào kho.
Hành Động (Lựa chọn 6)	1. **Nhập:** 6 2. **Hệ thống:** --- HÀNH ĐỘNG NÔNG TRẠI --- 3. **Hệ thống:** (Thông báo từ choTatCaAn) 4. **Hệ thống:** (Thông báo từ tuoiNuocTatCa)	Quay lại menu chính.

Bảng 4.6. Các trường hợp kiểm thử chức năng vận hành trang trại (Lưu Game, Thu Hoạch, Hành Động).

– Nâng cấp (lựa chọn 8)

Mục tiêu	Các bước thực hiện (Input & Output)	Kết quả cuối cùng
Nâng cấp Chuồng	1. **Nhập:** 8 2. **Hệ thống:** Hiển thị menu Nâng Cấp, Chọn: 3. **Nhập:** 1 4. **Hệ thống:** (Thông báo nâng cấp thành công) 5. **Hệ thống:** Trạng thái Chuồng mới: [0/X] (X >	Quay lại menu chính. Tiền bị trừ, sức chứa chuồng tăng.

	5)	
Lỗi (Không đủ tiền)	(Giả định Tiền < Chi phí nâng cấp) 1. **Nhập:** 8 2. **Hệ thống:** Chọn: 3. **Nhập:** 1 4. **Hệ thống:** (Thông báo lỗi "Không đủ tiền")	Quay lại menu chính. Tiền và sức chứa không đổi.
Quay lại (Lựa chọn 0)	1. **Nhập:** 8 2. **Hệ thống:** Chọn: 3. **Nhập:** 0	Quay lại menu chính. Không có gì thay đổi.
Lỗi (Nhập chữ)	1. **Nhập:** 8 2. **Hệ thống:** Chọn: 3. **Nhập:** abc 4. **Hệ thống:** [X] Lựa chọn không hợp lệ!	Quay lại menu chính.

Bảng 4.7. Kiểm thử nâng cấp công trình và xử lý lỗi tài chính

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Kết luận

Dựa trên các mục tiêu đã đề ra trong Lời mở đầu, dự án "Trò chơi nông trại" đã hoàn thành và đáp ứng đầy đủ các yêu cầu cốt lõi về mặt học thuật và kỹ thuật. Nhóm đã vận dụng thành công bốn nguyên lý nền tảng của lập trình hướng đối tượng (OOP) để xây dựng một hệ thống mô phỏng hoàn chỉnh trên giao diện dòng lệnh (CLI)

Các nguyên lý OOP đã áp dụng.

- Tính kế thừa: nhóm đã xây dựng được một cấu trúc lớp có tổ chức, bắt đầu từ lớp trừu tượng *SinhVat* và phân nhánh thành *DongVat, ThucVat*, sau đó mở rộng thành các lớp cụ thể như *Bo, Ga, Lua*. Cách thiết kế này giúp tái sử dụng mã nguồn hiệu quả, hạn chế trùng lặp và tạo sự dễ dàng khi mở rộng hệ thống
- Tính đa hình và sử dụng interface: dự án đã triển khai thành công các giao diện như *ThuHoach, NangCap và MuaBan*. Nhờ đó, các lớp như *Chuong, Vuon* hay *CuaHang* có thể sử dụng chung một tập phương thức mà không cần phụ thuộc vào loại đối tượng cụ thể, thể hiện rõ ràng lợi ích của đa hình trong lập trình hướng đối tượng
- Tính trừu tượng: lớp *NhiemVu* đã thể hiện vai trò làm khuôn mẫu chung cho các loại nhiệm vụ trong trò chơi. Phương thức *kiemTraHoanThanh()* được định nghĩa trừu tượng, yêu cầu các lớp con như *NhiemVuNangCap* hay *NhiemVuThuHoach* phải tự xây dựng logic kiểm tra phù hợp. Điều này giúp hệ thống nhiệm vụ linh hoạt và dễ dàng mở rộng.
- Tính đóng gói và hợp thành: lớp *NguoiChoi* là trung tâm điều phối, sở hữu các đối tượng *Chuong, Vuon* và *Kho*. Các lớp này được thiết kế để che giấu những xử lý bên trong như cách gộp sản phẩm trong *Kho*, giúp giao diện lập trình rõ ràng hơn và hạn chế sai sót khi thao tác dữ liệu.

Hướng phát triển

Để nâng cao trải nghiệm người dùng cũng như mở rộng chiều sâu mô phỏng, nhóm đề xuất một số hướng phát triển trong tương lai.

- Hệ thống sinh vật có thể được nâng cấp. Cụ thể, cần bổ sung các trạng thái như ốm, héo hoặc bị sâu bệnh. Những trạng thái này sẽ ảnh hưởng trực tiếp đến năng suất và khả năng thu hoạch, khiến việc chăm sóc trở nên quan trọng hơn thay vì chỉ thực hiện thao tác cho ăn hay tưới nước. Ngoài ra, nhóm có thể mô phỏng vòng đời sinh vật đầy đủ hơn, ví dụ: cây trồng trải qua các giai đoạn phát triển cho đến khi thu hoạch, và động vật có vòng đời cũng như tuổi thọ rõ ràng hơn.
- Logic trò chơi có thể được mở rộng theo hướng tăng tính quản lý tài nguyên. Ví dụ, có thể bổ sung hệ thống tiêu thụ thức ăn của động vật, yêu cầu người chơi phải duy trì lượng thức ăn trong kho. Bên cạnh đó, các yếu tố thời gian và sự kiện ngẫu nhiên như thời tiết, sâu bệnh hoặc các tình huống đặc biệt cũng có thể được thêm vào để tăng tính thử thách và sự sống động cho trò chơi.
- Giao diện trò chơi có thể tiếp tục được cải thiện. Ngoài việc cho phép người chơi tự chọn hoặc đặt tên file lưu trong lớp *XuLyFile*, nhóm có thể nghiên cứu chuyển từ giao diện dòng lệnh sang giao diện đồ họa dựa trên JavaScript kết hợp với HTML và CSS. Cách tiếp cận này sẽ mang lại trải nghiệm trực quan hơn, linh hoạt hơn và phù hợp với người chơi phổ thông.

TÀI LIỆU THAM KHẢO

[1] 2025-06, Eclipse IDE for Enterprise Java and Web Developers .

[2] Gemini.

[3] Visual Studio Code.

[4] Tài, B. A., & Tài, N. K. (2012). Được truy lục từ Xây dựng phần mềm trò chơi hỗ trợ học lập trình Pascal.