ĐẠI HỌC BÁCH KHOA HÀ NỘI

TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

──────── \* ───────

**BÀI TẬP LỚN**

MÔN: LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

**GAME CHƠI BÀI**

Nhóm : **N4**

Mã lớp học : 149475

Giáo viên hướng dẫn : Trịnh Tuấn Đạt

Danh sách sinh viên thực hiện:

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Họ tên** | **Mã sinh viên** | **Email** |
| 1 | Nguyễn Viết Văn | 20204861 | Van.NV204861@sis.hust.edu.vn |
| 2 | Dương Ngô Cung | 20225171 | cunggreen@gmail.com |
| 3 | Võ Minh Trí | 20210862 | tri.vm210862@sis.hust.edu.vn |
| 4 | Nguyễn Việt Anh | 20215307 | anh.nv215307@sis.hust.edu.vn |
| 5 | Đặng Duy Thịnh | 20204854 | Thinh.DD204854@sis.hust.edu.vn |
| 6 | Cao Nguyễn Thành Hoàng | 20224851 | Hoang.cnt224851@sis.hust.edu.vn |

***Hà Nội, tháng 12 năm 2024***

**MỤC LỤC**

[1.Chi tiết công việc 3](#_heading=h.30j0zll)

[1.1](#_heading=h.1fob9te) Phân công việc 3

[1.2. Tỷ lệ đóng góp 3](#_heading=h.3znysh7)

[2.Biểu đồ UML 4](#_heading=h.2et92p0)

[2.1. Sơ đồ ca sử dụng tổng quan 4](#_heading=h.tyjcwt)

[2.2 Biểu đồ phụ thuộc gói 5](#_heading=h.3dy6vkm)

[2.3 Biểu đồ lớp 5](#_heading=h.1t3h5sf)

[3. Class Documentation 7](#_heading=h.4d34og8)

[3.1 Capture 7](#_heading=h.2s8eyo1)

[3.2 Card 7](#_heading=h.17dp8vu)

[3.3 CardPane 9](#_heading=h.3rdcrjn)

[3.4 Combo 9](#_heading=h.26in1rg)

[3.5 Game 10](#_heading=h.lnxbz9)

[3.6 Pair 11](#_heading=h.35nkun2)

[3.7 Player 12](#_heading=h.1ksv4uv)

[3.8 Run 13](#_heading=h.44sinio)

[3.9 Straight 13](#_heading=h.2jxsxqh)

[3.10 Triple 14](#_heading=h.z337ya)

# 1.Chi tiết công việc

## Phân công việc

|  |  |
| --- | --- |
| Nguyễn Viết Văn | Tạo Gui |
| Dương Ngô Cung | Viết thuật toán cho Bot |
| Võ Minh Trí | Tạo các class chính |
| Nguyễn Việt Anh | Tạo Thuật cho game tiến lên |
| Đặng Duy Thịnh | Tạo thuật cho game 3 lá |
| Cao Nguyễn Thành Hoàng | Thiết kế giao diện, triển khai logic |

## 1.2. Tỷ lệ đóng góp

|  |  |
| --- | --- |
| Nguyễn Viết Văn | 5 |
| Dương Ngô Cung | 5 |
| Võ Minh Trí | 30 |
| Nguyễn Việt Anh | 5 |
| Đặng Duy Thịnh | 5 |
| Cao Nguyễn Thành Hoàng | 50 |

# 2.Biểu đồ UML

## 2.1. Sơ đồ ca sử dụng tổng quan

A diagram of a card game

Description automatically generated

2.2 Biểu đồ phụ thuộc góiA diagram of a computer system

Description automatically generated

## 2.3 Biểu đồ lớp

2.3.1 Biểu đồ lớp trong gói Module

A screenshot of a computer

Description automatically generated

2.3.1 Biểu đồ lớp trong gói gamelogic

A screenshot of a computer

Description automatically generated

# 3. Giải thích các lớp

# **3.1 Capture**

# Một lớp trừu tượng dùng để đại diện cho một kiểu bắt bài (capture) và tạo các kiểu bắt bài. Bao gồm các phương thức sau:

# **formCapture()** Phương thức trừu tượng, lớp con phải triển khai. Dùng để tạo bắt bài của loại tương ứng. Trả về đối tượng bắt bài nếu hợp lệ, null nếu không hợp lệ.

# **getScore()** Phương thức trừu tượng, lớp con phải triển khai. Dùng để lấy tổng điểm của kiểu bắt bài.

# **getCaptureName()** Phương thức trừu tượng, lớp con phải triển khai. Dùng để lấy tên của kiểu bắt bài.

# **getCaptureCard()** Lấy tất cả các lá bài tạo thành kiểu bắt bài này. Dùng bởi lớp Game để loại bỏ bài trong "pool" và hiển thị các kiểu bắt bài của người chơi.

# **formCaptures()** Một phương thức tĩnh. Tự động tạo kiểu bắt bài. Phương thức thử tạo các kiểu bắt bài khác nhau và trả về kiểu có điểm cao nhất. Trả về null nếu không tạo được kiểu bắt bài nào.

## 3.2 Card

# Lớp đại diện cho một lá bài, bao gồm các thuộc tính như hạng (rank), chất (suit), hình ảnh lá bài, và hình mặt sau của bài. Bao gồm các phương thức sau:

# Constructor Tạo đối tượng lá bài đại diện bởi giá trị từ 0 đến 51, mỗi số tượng trưng cho một lá bài trong bộ bài. Hạng và chất được tính toán.

# getSuit() Trả về chất dưới dạng chuỗi. Hữu ích để hiển thị và in ra bảng điều khiển.

# getSuitValue() Trả về giá trị đại diện cho chất. Hữu ích cho việc so sánh.

# getRank() Trả về hạng dưới dạng chuỗi. Hữu ích để hiển thị và in ra bảng điều khiển.

# getRankValue() Trả về giá trị đại diện cho hạng. Hữu ích để so sánh và tính toán.

# toString() Ghi đè phương thức toString() mặc định của Java để cung cấp chuyển đổi tốt hơn từ đối tượng sang chuỗi.

# newDeck() Trả về một mảng chứa bộ bài mới gồm 52 lá bài.

# shuffleDeck() Nhận vào một mảng bài và xáo trộn chúng ngẫu nhiên.

# compareTo() Triển khai giao diện Comparable để hỗ trợ sắp xếp tổng quát các đối tượng lá bài. Sắp xếp theo chất tăng dần, sau đó theo hạng tăng dần.

# getEmptyCard() Phương thức tĩnh trả về đối tượng lá bài đại diện cho 'không có gì'. Khác với null, vì null biểu thị lỗi, trong khi lá bài rỗng biểu thị không có lá bài.

# getImage() Trả về hình ảnh của lá bài. Hình ảnh của lá bài chỉ được tải khi cần để tăng hiệu suất.

# getBackImage() Phương thức tĩnh trả về hình mặt sau của lá bài. Tất cả các mặt sau đều giống nhau nên phương thức này là tĩnh.

## 3.3 CardPane

# Một "chỗ chứa" hình ảnh lá bài. Là lớp con của StackPane và có thêm chức năng hỗ trợ lớp chính trong xử lý sự kiện. Cấu trúc: StackPane -> ImageView -> Image. Bao gồm các phương thức:

# Constructor Tạo ImageView, thiết lập chiều rộng và các thuộc tính, sau đó đặt ImageView vào StackPane.

# setCard() Gắn đối tượng lá bài với CardPane và gắn trình xử lý sự kiện (chọn hoặc bỏ chọn lá bài).

# setCurrentPlayer() Thiết lập ngữ cảnh cho người chơi hiện tại. Tất cả các hành động như chọn hoặc bỏ chọn bài sẽ do người chơi hiện tại thực hiện.

# setActive() Thiết lập trạng thái hoạt động của lá bài. Lá bài hoạt động có thể chọn hoặc bỏ chọn, trong khi lá bài không hoạt động thì không thể.

## 3.4 Combo

# Lớp đại diện cho kiểu bắt bài loại Combo. Bao gồm các phương thức:

# public no-args Constructor Tạo đối tượng không có kiểu bắt bài để có thể sử dụng phương thức formCapture().

# private Constructor Dùng bởi formCapture() để tạo đối tượng bắt bài hợp lệ. Constructor này là private để ngăn chặn tạo đối tượng bắt bài không hợp lệ.

# formCapture() Kiểm tra xem tổ hợp lá bài có phải là kiểu bắt bài hợp lệ không. Nếu hợp lệ, trả về đối tượng Combo, nếu không thì trả về null.

# getScore() Trả về điểm số của kiểu bắt bài.

# getCaptureName() Trả về tên của kiểu bắt bài dưới dạng chuỗi.

## 3.5 Game

# Lớp chính của toàn bộ chương trình Java. Chịu trách nhiệm thực hiện logic trò chơi và thiết kế giao diện đồ họa (GUI). Bao gồm các phương thức:

# setNextPlayer() Hàm trợ giúp để chuyển ngữ cảnh sang người chơi tiếp theo.

# initializeRound() Hàm trợ giúp để khởi tạo một vòng chơi mới. Bộ bài được xáo trộn và phân phát cho "pool" và các người chơi.

# formCaptureImage() Hàm trợ giúp để tạo danh sách hình ảnh kiểu bắt bài ở bảng bên phải. Kết quả có thể được thêm trực tiếp vào bảng bên phải.

# start() Hàm chính của chương trình. Thiết kế giao diện GUI ở phần đầu, sau đó gắn các trình xử lý sự kiện cho từng thành phần. Hai loại sự kiện chính:

# Sự kiện chọn số người chơi.

# Sự kiện nút bắt bài. Xử lý sự kiện chịu trách nhiệm kiểm tra kiểu bắt bài hợp lệ, cập nhật điểm số, bảng bên phải và thông báo người thắng.

# main() Khởi chạy ứng dụng JavaFX.SS

# (Tương tự, các lớp Pair, Player, Run, Straight, và Triple có mô tả gần giống lớp Combo, điều chỉnh phù hợp theo từng kiểu bắt bài).

## 3.6 Class Tienlen: Kế thừa từ lớp Game với các phương thức

1. isStraight

**Chức năng**:  
Kiểm tra xem một tập hợp các lá bài có tạo thành một **dãy liên tiếp** hay không.  
**Quy tắc**:

* Tập hợp phải có ít nhất 3 lá bài.
* Các lá bài phải liên tiếp về giá trị (ví dụ: 3-4-5 hoặc 10-J-Q).
* Lá bài 2 không được phép xuất hiện trong dãy (theo luật thông thường của Tiến Lên).

**Cách hoạt động**:

* Sắp xếp tập hợp các lá bài theo thứ tự tăng dần.
* Kiểm tra giá trị liên tiếp giữa các lá bài.
* Nếu tất cả các lá bài hợp lệ và liên tiếp, trả về true, ngược lại trả về false.

**2. isDouble**

**Chức năng**:  
Kiểm tra xem một tập hợp các lá bài có tạo thành một **đôi** hay không.  
**Quy tắc**:

* Tập hợp phải chứa chính xác 2 lá bài.
* Hai lá bài phải cùng một giá trị (ví dụ: đôi 7, đôi J).

**Cách hoạt động**:

* Kiểm tra kích thước của tập hợp (phải là 2 lá).
* So sánh giá trị của hai lá bài, nếu giống nhau, trả về true, ngược lại trả về false.

3. isTriple

**Chức năng**:  
Kiểm tra xem một tập hợp các lá bài có tạo thành một **bộ ba** hay không.  
**Quy tắc**:

* Tập hợp phải chứa chính xác 3 lá bài.
* Ba lá bài phải cùng một giá trị (ví dụ: bộ ba 9, bộ ba Q).

**Cách hoạt động**:

* Kiểm tra kích thước của tập hợp (phải là 3 lá).
* So sánh giá trị của cả ba lá bài, nếu giống nhau, trả về true, ngược lại trả về false.

4. isQuara

**Chức năng**:  
Kiểm tra xem một tập hợp các lá bài có tạo thành một **tứ quý** hay không.  
**Quy tắc**:

* Tập hợp phải chứa chính xác 4 lá bài.
* Bốn lá bài phải cùng một giá trị (ví dụ: tứ quý 5, tứ quý K).

**Cách hoạt động**:

* Kiểm tra kích thước của tập hợp (phải là 4 lá).
* So sánh giá trị của tất cả bốn lá bài, nếu giống nhau, trả về true, ngược lại trả về false.

## 3.6 Class 3 lá

**1. Phương thức value()**

**Chức năng**:  
Tính điểm số (value) của một bộ bài gồm 3 lá, để so sánh hoặc xác định độ mạnh yếu giữa các bộ bài.

**Quy tắc tính điểm**:

* **Điểm chính (Main Value)**:  
  Điểm chính là giá trị của các lá bài (ví dụ: A = 14, K = 13, Q = 12, J = 11, ..., 3 = 3).
* **Điểm phụ (Suit Priority)**:  
  Nếu hai bộ có giá trị chính giống nhau, ưu tiên lá bài có chất mạnh hơn. Quy ước chất theo thứ tự:  
  **Bích (♠) > Cơ (♥) > Rô (♦) > Tép (♣)**.
* **Cách tính tổng giá trị**:
  + Tổng giá trị có thể là tổng điểm của các lá bài hoặc một công thức tổ hợp dựa trên điểm và chất để đảm bảo duy nhất.

**2. Phương thức isStronger()**

**Chức năng**:  
So sánh độ mạnh giữa hai bộ **Ba Lá**, xác định bộ nào mạnh hơn.

**Quy tắc so sánh**:

1. So sánh giá trị tổng của hai bộ (dựa vào phương thức value()).
2. Nếu giá trị tổng bằng nhau, so sánh chất của lá bài lớn nhất trong mỗi bộ.

# 4. Tối ưu hóa phần mềm

## 4.1 Sử dụng một lớp scene duy nhất cho cả 2 game bằng lập trình

## tổng quát.

1. **Tái sử dụng mã nguồn (Code Reusability):**

* Một lớp Scene duy nhất có thể được sử dụng cho nhiều trò chơi khác nhau hoặc nhiều chế độ trong cùng một trò chơi.
* Hạn chế trùng lặp mã bằng cách chỉ cần thay đổi logic trò chơi thông qua việc tạo các lớp kế thừa khác nhau từ TienLenGame.

1. **Dễ bảo trì (Maintainability):**

* Khi cần cập nhật giao diện hoặc logic chung (ví dụ: xử lý đầu vào, hiển thị màn hình), chỉ cần thực hiện thay đổi tại một vị trí duy nhất.
* Việc kiểm tra và sửa lỗi trở nên dễ dàng hơn nhờ phạm vi ảnh hưởng được thu hẹp.

1. **Tối ưu hóa bộ nhớ và hiệu suất:**

* Thay vì tạo nhiều lớp Scene riêng biệt với logic tương tự, sử dụng một lớp duy nhất giúp tiết kiệm tài nguyên.
* Xử lý sự kiện được thống nhất, giảm thiểu rủi ro lặp lại các công việc giống nhau.

1. **Tính tổng quát và mở rộng (Generalization and Extensibility):**

* Khi bổ sung chế độ chơi mới (ví dụ: "3 Lá"), chỉ cần tạo một lớp mới kế thừa từ TienLenGame mà không cần chỉnh sửa lớp Scene.

## 4.2 Áp dụng lớp trừu tượng Game:

* **Tuân thủ nguyên tắc SOLID (Open/Closed Principle):**  
  Lớp Game đảm bảo rằng các trò chơi cụ thể chỉ cần kế thừa và triển khai chi tiết mà không làm thay đổi mã gốc. Điều này giúp hệ thống dễ dàng mở rộng và bảo trì, giữ cho mã nguồn ổn định và linh hoạt.
* **Tăng cường tính trừu tượng và linh hoạt (Abstraction and Flexibility):**  
  Lớp Scene chỉ cần tương tác với Game mà không cần quan tâm đến chi tiết, từ đó nâng cao khả năng tái sử dụng và làm cho mã nguồn dễ dàng thích ứng với các thay đổi.
* **Khả năng mở rộng trò chơi:**
* Việc bổ sung chế độ chơi mới hoặc các biến thể khác chỉ cần tạo một lớp con kế thừa từ Game, mà không ảnh hưởng đến các phần khác của hệ thống.
* Các chức năng chung (như chia bài, kiểm tra lượt chơi) được tái sử dụng thông qua kế thừa, giảm thiểu viết lại mã.
* **Phân tách trách nhiệm rõ ràng (Separation of Concerns):**
* Lớp Scene: Quản lý giao diện người dùng và xử lý tương tác.
* Lớp Game: Xử lý logic trò chơi cụ thể.
* Việc tách biệt này giúp hệ thống dễ hiểu hơn, giảm sự phụ thuộc giữa các thành phần, và nâng cao tính bảo trì.
* **Cải thiện khả năng kiểm thử (Testability):**
* Các lớp con như TienLen và 3 lá có thể được kiểm thử độc lập để đảm bảo luật chơi được thực hiện chính xác mà không làm ảnh hưởng đến lớp Scene.
* Điều này giúp dễ dàng phát hiện và sửa lỗi trong các thành phần riêng lẻ.

4.Giải thích các packets  
4.1. Gói module:

Module được tạo ra để chứa các lớp Card, Deck, Player, Bot

4.2 Gói gamelogic:

Chứa các lớp TienLen, Game, 3 Lá có nhiệm vụ xử lí logic, luật chơi của trò chơi.

## 4.3. Gói gamescene

Định nghĩa và quản lí các giao diện của trò chơi. Cung cấp các

màn hình tương tác để người chơi chọn loại game, nhập thông tin và

màn. Kết nối với chặt chẽ với logic để hiển thị chính xác trạng thái của

trò chơi

## 4.4 Gói imageaction

Gói imageaction chứa các lớp (class) liên quan đến việc xử lý và

hiển thị các hình ảnh trong giao diện người dùng của game. chịu trách

nhiệm chính trong việc cung cấp hình ảnh minh họa và cải thiện trải

nghiệm thị giác của người chơi. Các lớp trong gói này tập trung vào việc

tạo và quản lý các thành phần hình ảnh, giúp giao diện trở nên sinh

động và rõ ràng hơn.

4.5. Gói controller

Có nhiệm vụ quản lý và tương tác giữa giao diện người dùng

(được định nghĩa trong các file .fxml) với logic game. Các lớp trong gói

này đảm bảo sự đồng bộ giữa các thay đổi trong giao diện và trạng thái

của game.

4.6. Gói application

Chứa lớp Main, có nhiệm vụ khởi tạo màn hình ban đầu và chạy

chương trình.

# 5. Giải thích các kỹ thuật lập trình hướng đối tượng đã áp dụng

## Trừu tượng hóa

* + Sử dụng enum để định nghĩa giá trị (Value) và chất bài (Type), giúp mã dễ hiểu, tránh lỗi, và tăng tính mô tả.

## Kế thừa

* + **Player** kế thừa từ **Deck**, cho phép người chơi thao tác trên bộ bài (thêm, xóa, in bài) mà không cần viết lại mã.
  + **Bot** kế thừa từ **Player**, thêm logic riêng cho bot, hỗ trợ mở rộng mà không thay đổi lớp cơ sở.
  + **TienLen** và **3 lá** kế thừa từ **Game**, giảm lặp mã, hỗ trợ mở rộng tính năng mà không ảnh hưởng lớp cơ sở.

## Đa hình

* + Phương thức getCard trong **Player** hoạt động với nhiều loại game khác nhau nhờ sử dụng kiểu tổng quát <T extends TienLen>.
  + Tăng tính linh hoạt, giảm sự phụ thuộc vào một loại game cụ thể.

## Ghi đè phương thức (Override)

* + toString() trong **Card**: Trả về chuỗi mô tả lá bài, hỗ trợ debug hoặc hiển thị giao diện.
  + clone() trong **Deck**: Tạo bản sao bộ bài, tiện lợi khi chơi nhiều ván.

## Xử lý ngoại lệ (Exception Handling)

* + **Player** sử dụng try-catch trong phương thức chơi bài, ngăn lỗi khi truy cập bộ bài, đảm bảo chương trình ổn định và thân thiện với người dùng.

## Kết tập (Composition)

* + **Deck** chứa các **Card**, nhưng **Card** tồn tại độc lập, dễ quản lý (thêm, xóa, sắp xếp) mà không ảnh hưởng lớp **Card**.
  + **Player** sở hữu **Deck** thông qua kết tập: Bộ bài tồn tại riêng biệt, dễ tái sử dụng khi người chơi bị loại.
  + **BOT** thao tác với **TienLen**qua logic riêng mà không phụ thuộc chặt chẽ, giúp dễ mở rộng sang các game khác.

## Lập trình tổng quát (Generic Programming)

* + Sử dụng <T extends TienLen> trong các lớp như **3 Lá** để xử lý đa dạng game, tăng tính linh hoạt, giảm phức tạp.

## Đóng gói (Encapsulation)

* + Các thuộc tính (handCards, choosenCards) được khai báo protected hoặc private, ẩn chi tiết triển khai.
  + Các phương thức logic được đóng gói, chỉ cung cấp API cần thiết, bảo mật và dễ quản lý trạng thái đối tượng.