Đối với yêu cầu a, để đưa trò chơi LanguageGame vào hệ thống trò chơi hiện tại và chạy phương thức run(), có thể áp dụng mẫu thiết kế Strategy Pattern.

Strategy Pattern là một mẫu thiết kế hành vi (behavioral pattern) trong đó các đối tượng được tách ra thành các thuật toán riêng biệt, có thể tùy chọn được sử dụng trong quá trình chạy. Mẫu thiết kế này giúp cho việc tách rời các thuật toán khác nhau khỏi đối tượng chứa nó, giúp dễ dàng thay đổi thuật toán tại thời điểm thực thi.

Áp dụng mẫu thiết kế này vào trường hợp này, ta có thể tạo ra một interface GameStrategy chứa phương thức run(), và tạo ra các class implement interface này cho từng trò chơi cụ thể. Với trò chơi LanguageGame, ta tạo ra một class LanguageGameStrategy implement GameStrategy và định nghĩa lại phương thức run() theo nội dung của trò chơi đó. Sau đó, ta sử dụng mẫu Factory Method để tạo ra đối tượng của class LanguageGameStrategy tại thời điểm thực thi khi cần chạy trò chơi LanguageGame.

Đối với yêu cầu b, ta có thể áp dụng mẫu thiết kế Template Method Pattern để định nghĩa các bước thực hiện của trò chơi và để cho phép lớp con có thể mở rộng và định nghĩa lại phần hướng dẫn và phần tổng hợp, nhận xét.

Template Method Pattern là một mẫu thiết kế hành vi (behavioral pattern) cho phép định nghĩa một bộ khung (template) cho các thuật toán cụ thể. Mẫu thiết kế này giúp cho việc giữ nguyên cấu trúc chung của các thuật toán, nhưng cho phép các lớp con có thể mở rộng và định nghĩa lại các phần cụ thể của thuật toán.

a. Để đưa trò chơi LanguageGame vào hệ thống trò chơi hiện tại và thực hiện phương thức run() của nó, chúng ta có thể sử dụng mẫu thiết kế Facade Pattern. Mẫu thiết kế này cho phép chúng ta định nghĩa một lớp đại diện (facade) để cung cấp một giao diện thống nhất cho việc sử dụng các thành phần phức tạp trong hệ thống.

Với trường hợp này, chúng ta có thể tạo ra một lớp gọi là GameFacade để đại diện cho hệ thống trò chơi và cung cấp một phương thức đơn giản là startGame(), khi được gọi, nó sẽ tạo ra một đối tượng của lớp LanguageGame và gọi phương thức run() của nó để bắt đầu trò chơi.

b. Để bổ sung phần hướng dẫn và phần tổng hợp, nhận xét về quá trình chơi trò chơi, chúng ta có thể sử dụng mẫu thiết kế Decorator Pattern. Mẫu thiết kế này cho phép chúng ta bổ sung các chức năng mới cho một đối tượng mà không thay đổi cấu trúc của đối tượng đó.

Với trường hợp này, chúng ta có thể tạo ra một lớp gọi là GameDecorator, đại diện cho tất cả các trò chơi trong hệ thống. Lớp này sẽ bổ sung hai phương thức mới là showInstruction() và showSummary() để hiển thị phần hướng dẫn và phần tổng hợp sau khi trò chơi kết thúc.

Ngoài ra, chúng ta cũng có thể tạo ra các lớp con của GameDecorator, ví dụ như LanguageGameDecorator, để bổ sung các chức năng cụ thể cho từng trò chơi. Ví dụ, lớp LanguageGameDecorator có thể bổ sung chức năng hiển thị các câu hỏi trong trò chơi ngôn ngữ.

Khi thực hiện, chúng ta có thể tạo một đối tượng của lớp LanguageGameDecorator, truyền vào một đối tượng của lớp LanguageGame để kết hợp các chức năng của hai lớp này lại với nhau. Khi trò chơi kết thúc, chúng ta có thể gọi phương thức showSummary() của đối tượng LanguageGameDecorator để hiển thị phần tổng hợp, nhận xét về quá trình chơi trò chơi đó.

b) Để xử lý yêu cầu này, ta có thể áp dụng mẫu thiết kế Decorator và Template Method.

Mẫu Decorator:

Mẫu Decorator cho phép ta thêm các tính năng mới cho đối tượng hiện tại mà không cần phải sửa đổi đối tượng ban đầu. Với yêu cầu này, ta có thể sử dụng Decorator để thêm phần hướng dẫn và phần nhận xét cho trò chơi.

Tạo một interface IGame chứa phương thức run() để thực hiện game.

Tạo một lớp AbstractGameDecorator kế thừa IGame và cũng chứa phương thức run(). Lớp này sẽ là lớp trừu tượng để thêm tính năng cho trò chơi.

Tạo các lớp ConcreteGameDecorator kế thừa AbstractGameDecorator và cài đặt phương thức run() để thêm tính năng hướng dẫn và nhận xét.

Mẫu Template Method:

Mẫu Template Method cho phép ta xác định khuôn mẫu chung cho một thuật toán và cho phép lớp con triển khai lại các bước cụ thể của thuật toán đó. Với yêu cầu này, ta có thể sử dụng Template Method để xác định cấu trúc chung cho các trò chơi và cho phép các lớp con triển khai các bước cụ thể của trò chơi.

Tạo một lớp AbstractGame chứa một phương thức playGame() là phương thức Template Method để thực hiện trò chơi.

Tạo các lớp con kế thừa AbstractGame và triển khai lại phương thức playGame() để thực hiện các bước cụ thể của trò chơi.

Tạo các lớp Decorator kế thừa AbstractGame và sử dụng Decorator để thêm tính năng hướng dẫn và nhận xét cho trò chơi.

Khi sử dụng cả hai mẫu thiết kế này, ta có thể tạo ra các lớp trò chơi mới và thêm tính năng hướng dẫn và nhận xét cho chúng mà không cần phải sửa đổi các lớp trò chơi ban đầu. Ngoài ra, ta cũng có thể dễ dàng mở rộng và bảo trì hệ thống trò chơi khi có nhu cầu.

Đối với yêu cầu a, để đưa trò chơi LanguageGame vào hệ thống trò chơi hiện tại và chạy phương thức run(), có thể áp dụng mẫu thiết kế Strategy Pattern.

Strategy Pattern là một mẫu thiết kế hành vi (behavioral pattern) trong đó các đối tượng được tách ra thành các thuật toán riêng biệt, có thể tùy chọn được sử dụng trong quá trình chạy. Mẫu thiết kế này giúp cho việc tách rời các thuật toán khác nhau khỏi đối tượng chứa nó, giúp dễ dàng thay đổi thuật toán tại thời điểm thực thi.

Áp dụng mẫu thiết kế này vào trường hợp này, ta có thể tạo ra một interface GameStrategy chứa phương thức run(), và tạo ra các class implement interface này cho từng trò chơi cụ thể. Với trò chơi LanguageGame, ta tạo ra một class LanguageGameStrategy implement GameStrategy và định nghĩa lại phương thức run() theo nội dung của trò chơi đó. Sau đó, ta sử dụng mẫu Factory Method để tạo ra đối tượng của class LanguageGameStrategy tại thời điểm thực thi khi cần chạy trò chơi LanguageGame.

Đối với yêu cầu b, ta có thể áp dụng mẫu thiết kế Template Method Pattern để định nghĩa các bước thực hiện của trò chơi và để cho phép lớp con có thể mở rộng và định nghĩa lại phần hướng dẫn và phần tổng hợp, nhận xét.

Template Method Pattern là một mẫu thiết kế hành vi (behavioral pattern) cho phép định nghĩa một bộ khung (template) cho các thuật toán cụ thể. Mẫu thiết kế này giúp cho việc giữ nguyên cấu trúc chung của các thuật toán, nhưng cho phép các lớp con có thể mở rộng và định nghĩa lại các phần cụ thể của thuật toán.

a. Để đưa trò chơi LanguageGame vào hệ thống trò chơi hiện tại và thực hiện phương thức run() của nó, chúng ta có thể sử dụng mẫu thiết kế Facade Pattern. Mẫu thiết kế này cho phép chúng ta định nghĩa một lớp đại diện (facade) để cung cấp một giao diện thống nhất cho việc sử dụng các thành phần phức tạp trong hệ thống.

Với trường hợp này, chúng ta có thể tạo ra một lớp gọi là GameFacade để đại diện cho hệ thống trò chơi và cung cấp một phương thức đơn giản là startGame(), khi được gọi, nó sẽ tạo ra một đối tượng của lớp LanguageGame và gọi phương thức run() của nó để bắt đầu trò chơi.

b. Để bổ sung phần hướng dẫn và phần tổng hợp, nhận xét về quá trình chơi trò chơi, chúng ta có thể sử dụng mẫu thiết kế Decorator Pattern. Mẫu thiết kế này cho phép chúng ta bổ sung các chức năng mới cho một đối tượng mà không thay đổi cấu trúc của đối tượng đó.

Với trường hợp này, chúng ta có thể tạo ra một lớp gọi là GameDecorator, đại diện cho tất cả các trò chơi trong hệ thống. Lớp này sẽ bổ sung hai phương thức mới là showInstruction() và showSummary() để hiển thị phần hướng dẫn và phần tổng hợp sau khi trò chơi kết thúc.

Ngoài ra, chúng ta cũng có thể tạo ra các lớp con của GameDecorator, ví dụ như LanguageGameDecorator, để bổ sung các chức năng cụ thể cho từng trò chơi. Ví dụ, lớp LanguageGameDecorator có thể bổ sung chức năng hiển thị các câu hỏi trong trò chơi ngôn ngữ.

Khi thực hiện, chúng ta có thể tạo một đối tượng của lớp LanguageGameDecorator, truyền vào một đối tượng của lớp LanguageGame để kết hợp các chức năng của hai lớp này lại với nhau. Khi trò chơi kết thúc, chúng ta có thể gọi phương thức showSummary() của đối tượng LanguageGameDecorator để hiển thị phần tổng hợp, nhận xét về quá trình chơi trò chơi đó.

Đối với yêu cầu a, để đưa trò chơi LanguageGame vào hệ thống trò chơi hiện tại và chạy phương thức run(), có thể áp dụng mẫu thiết kế Strategy Pattern.

Strategy Pattern là một mẫu thiết kế hành vi (behavioral pattern) trong đó các đối tượng được tách ra thành các thuật toán riêng biệt, có thể tùy chọn được sử dụng trong quá trình chạy. Mẫu thiết kế này giúp cho việc tách rời các thuật toán khác nhau khỏi đối tượng chứa nó, giúp dễ dàng thay đổi thuật toán tại thời điểm thực thi.

Áp dụng mẫu thiết kế này vào trường hợp này, ta có thể tạo ra một interface GameStrategy chứa phương thức run(), và tạo ra các class implement interface này cho từng trò chơi cụ thể. Với trò chơi LanguageGame, ta tạo ra một class LanguageGameStrategy implement GameStrategy và định nghĩa lại phương thức run() theo nội dung của trò chơi đó. Sau đó, ta sử dụng mẫu Factory Method để tạo ra đối tượng của class LanguageGameStrategy tại thời điểm thực thi khi cần chạy trò chơi LanguageGame.

Đối với yêu cầu b, ta có thể áp dụng mẫu thiết kế Template Method Pattern để định nghĩa các bước thực hiện của trò chơi và để cho phép lớp con có thể mở rộng và định nghĩa lại phần hướng dẫn và phần tổng hợp, nhận xét.

Template Method Pattern là một mẫu thiết kế hành vi (behavioral pattern) cho phép định nghĩa một bộ khung (template) cho các thuật toán cụ thể. Mẫu thiết kế này giúp cho việc giữ nguyên cấu trúc chung của các thuật toán, nhưng cho phép các lớp con có thể mở rộng và định nghĩa lại các phần cụ thể của thuật toán.