Tên: Lê Trường Huy

MSV:5851071030

**Design Patterns Elements of Reusable Object-Oriented Software**

* **Summary:**

Vấn đề thiết kế

Chúng ta sẽ xem xét vấn đề trong lexi: **1**. Cấu trúc tài liệu **2**.Định dạng **3**.Hoàn thiện giao diện người dùng **4**. Hỗ trợ nhiều giao diện **5**.Hoạt động người dùng **6**.Kiểm tra chính tả **7**.Cấu trúc tài liệu

1. Cấu trúc tài liệu:

Một tài liệu cuối cùng chỉ là một sự sắp xếp của các thành phần đồ họa cơ bản như ký tự, đường thẳng, đa giác và các hình dạng khác. Các yếu tố này nắm bắt tổng số nội dung thông tin của tài liệu. Tuy nhiên, anauthor thường xem những yếu tố này không trong thuật ngữ đồ họa nhưng trong thuật ngữ của cấu trúc vật lý của tài liệu — dòng, cột, hình, bảng và các cấu trúc con khác. Đổi lại, các cấu trúc con này có cấu trúc của chúng, v.v.

Ngoài những mục tiêu này là một số ràng buộc. Đầu tiên, chúng ta nên tạo văn bản và đồ họa đồng nhất. Giao diện của ứng dụng cho phép người dùng nhúng văn bản vào đồ họa tự do và ngược lại. Chúng ta không nên coi đồ họa là một thứ đặc biệt trường hợp của văn bản hoặc văn bản như một chữ cái đặc biệt của đồ họa; nếu không chúng ta sẽ kết thúc với cơ chế định dạng và điều chỉnh dự phòng. Một bộ cơ chế nên đủ cho văn bản và đồ họa.

Thứ hai, việc triển khai của chúng tôi không cần phải phân biệt các phần tử đơn lẻ và các nhóm phần tử trong biểu diễn bên trong. exi sẽ có thể xử lý các yếu tố đơn giản và phức tạp đồng nhất, do đó cho phép tùy ý phức tạp các tài liệu. Ví dụ, phần phụ ở dòng năm của cột hai có thể là một ký tự đơn hoặc một sơ đồ phức tạp với nhiều thành phần phụ. Miễn là chúng ta biết phần tử này có thể tự vẽ và chỉ định kích thước của nó, tính đơn giản của nó không có về cách thức và vị trí nó sẽ xuất hiện trên trang.

2.3 Định dạng

Quá trình định dạng, với tất cả các ràng buộc và chi tiết của nó, không hề dễ dàng tự động hóa, người ta đã đưa ra nhiều thuật toán định dạng với các điểm mạnh yếu khác nhau. Các thuật toán định dạng có xu hướng phức tạp tốt hơn hết giữ nó đọc lập với cấu trúc tài liệu. Thiết kế Lexi sao cho dễ thay đổi thuật toán. Chúng ta có thể cô lập thuật toán và làm cho thuật toán có thể thay thế dễ dàng bằng cách gói vào 1 đối tượng.

2.4 Hoàn thiện giao diện người dùng

Thành phần đối tượng cung cấp khả năng linh hoạt và khả thi hơn cơ chế. Chúng ta có thể áp dụng khái niệm bao vây trong suốt cho tất cả các glyphs khác. chúng tôi sẽ xác định lớp dưới cùng của Glyph được gọi là MonoGlyph để phục vụ như một lớp trừu tượng cho "embellishment glyphs"

MonoGlyph lưu trữ một tham chiếu đến một thành phần và chuyển tiếp tất cả yêu cầu nó. Điều đó làm cho MonoGlyph hoàn toàn minh bạch với khách hàng theo mặc định.

2.5 Hỗ trợ đa tiêu chuẩn Look-and-Feel

Đạt được tính di động trên các nền tảng phần cứng và phần mềm là một vấn đề nan giải trong thiết kế hệ thống. Chúng ta nên chuyển dễ dàng như có thể.

Mọi thứ chúng ta thấy và tương tác trong giao diện người dùng của Lexi là một glyph được tạo thành trong các glyph vô hình khác như hàng và cột. Các glyph hữu hình có thể nhìn thấy được những cái như nút và ký tự và sắp xếp hợp lý.

Factory Method là một mẫu thiết kế sáng tạo cung cấp một giao diện để tạo các đối tượng trong lớp cha, nhưng cho phép các lớp con thay đổi loại đối tượng sẽ được tạo.

Abstract Factory là một mẫu thiết kế sáng tạo cho phép bạn tạo ra các họ các đối tượng liên quan mà không cần chỉ định các lớp cụ thể của chúng.

2.6 Hỗ trợ đa cửa sổ hệ thống

Nó quản lý không gian màn hình cho các cửa sổ và định tuyến đầu vào cho chúng từ bàn phím và chuột. Một số hệ thống cửa sổ quan trọng và phần lớn không tương thích tồn tại ngày nay. Chúng tôi muốn Lexi chạy càng nhiều càng tốt vì những lý do chính xác giống như chúng tôi hỗ trợ nhiều tiêu chuẩn giao diện.

Thoạt nhìn, điều này có thể giống như một cơ hội khác để áp dụng mẫu Abstract Factory. Nhưng những hạn chế đối với tính di động của hệ thống cửa sổ khác đáng kể so với những hạn chế đối với sự độc lập về giao diện.

Chúng tôi đã giải quyết vấn đề này cho các widget bằng cách phát triển các lớp sản phẩm cụ thể và trừu tượng của riêng mình.

Lớp Window đóng gói những thứ mà cửa sổ có xu hướng làm trên các hệ thống cửa sổ:

- Chúng cung cấp các thao tác để vẽ các hình dạng hình học cơ bản

- Họ có thể tự biểu tượng hóa và khử biểu tượng.

- Họ có thể tự thay đổi kích thước.

- Họ có thể (lại) vẽ nội dung của mình theo yêu cầu

Lớp Window phải mở rộng chức năng của các cửa sổ từ các hệ thống cửa sổ khác nhau. Hãy xem xét hai triết lý cực đoan:

- Giao điểm của chức năng. Giao diện lớp Window chỉ cung cấp chức năng chung cho tất cả các hệ thống cửa sổ. Vấn đề với cách tiếp cận này là giao diện Window của chúng tôi chỉ mạnh bằng hệ thống cửa sổ kém khả năng nhất.

- Liên minh các chức năng. Tạo một giao diện kết hợp các khả năng của tất cả các hệ thống hiện có. Vấn đề ở đây là giao diện kết quả có thể rất lớn và không mạch lạc.

2.7 Hoạt động của người dùng

Điều còn thiếu là một cơ chế cho phép chúng tôi tham số hóa các mục menu theo yêu cầu mà chúng phải đáp ứng. Bằng cách đó, chúng tôi tránh được sự gia tăng của các lớp con và cho phép linh hoạt hơn trong thời gian chạy. Chúng tôi có thể tham số hóa MenuItem bằng một hàm để gọi, nhưng đó không phải là giải pháp hoàn chỉnh vì ít nhất ba lý do:

* Nó không giải quyết vấn đề hoàn tác / làm lại.
* Thật khó để liên kết trạng thái với một hàm.
* Các chức năng khó mở rộng và khó sử dụng lại các phần của chúng.

Command là một mẫu thiết kế hành vi biến một yêu cầu thành một đối tượng độc lập chứa tất cả thông tin về yêu cầu. Việc chuyển đổi này cho phép bạn tham số hóa các phương thức với các yêu cầu khác nhau, trì hoãn hoặc xếp hàng đợi việc thực hiện yêu cầu và hỗ trợ các hoạt động hoàn tác.

Vì vậy, để xác định xem một lệnh có thể hoàn tác hay không, chúng ta thêm một thao tác Có thể đảo ngược trừu tượng vào giao diện Command. Có thể đảo ngược trả về giá trị Boolean. Các lớp con có thể xác định lại thao tác này để trả về true hoặc false dựa trên tiêu chí thời gian chạy.

2.8 Kiểm tra chính tả và gạch nối

Vấn đề thiết kế cuối cùng liên quan đến phân tích văn bản, đặc biệt kiểm tra lỗi chính tả và giới thiệu các điểm gạch nối nếu cần để định dạng tốt.

Mẫu Iterator nắm bắt các kỹ thuật này để hỗ trợ truy cập và duyệt qua các cấu trúc đối tượng. Nó không chỉ áp dụng cho các cấu trúc tổng hợp mà còn cho các bộ sưu tập. Nó tóm tắt thuật toán duyệt và bảo vệ các máy khách khỏi cấu trúc bên trong của các đối tượng mà chúng đi qua.

* ***Ưu điểm:***

Mẫu có thể tái sử dụng trong nhiều dự án, cung cấp các giải pháp giúp xác định

kiến trúc hệ thống. Mẫu nắm bắt những kinh nghiệm kỹ thuật phần mềm, cung cấp sự

minh bạch cho việc thiết kế một ứng dụng. Mẫu là những giải pháp đã được chứng

minh và chứng thực vì chúng được xây dựng dựa trên kiến thức và kinh nghiệm của

các nhà chuyên gia phát triển phần mềm. Các mẫu thiết kế không đảm bảo một giải

pháp tuyệt đối cho một vấn đề. Chúng cung cấp sự rõ ràng cho kiến trúc hệ thống và

khả năng xây dựng một hệ thống tốt hơn

* ***Nhược điểm:***

Dù thiết kế mẫu đem lại nhiều lợi ích trong phát triển phần mềm. Tuy nhiên việc

sử dụng mẫu cũng còn tùy thuộc vào từng tình huống cụ thể trong từng dự án cụ thể.

Nhược điểm của mẫu cũng bộc lộ qua một số tính huống sau đây:

* Việc sử dụng quá nhiều mẫu cũng như buộc chúng phải phù hợp với chương

trình sẽ làm cho các đoạn mã trở nên rắc rối và khó hiểu hơn.

* Không có một phương pháp phát triển phần mềm nào là hoàn thiện và Design

Pattern không phải là một ngoại lệ.

* Không thích hợp cho những lập trình viên còn ít kinh nghiệm cũng như chưa

hiểu hết về Design Pattern mà áp dụng vào trong chương trình.