**Advice on Implementing Design Patterns for JabberPoint**

*Objective:*

To improve the maintainability, extensibility, and modularity of JabberPoint, several design patterns should be implemented. These patterns will address tight coupling, code duplication, and lack of flexibility in the system.

**Summary of Recommended Patterns**

|  |  |  |
| --- | --- | --- |
| **Design Pattern** | **Purpose** | **Why It’s Needed?** |
| **Factory Method** | Standardizes object creation | Prevents scattered new object creation |
| **Composite** | Manages hierarchical slide elements | Allows slides to contain nested elements |
| **Command** | Encapsulates user actions | Standardizes keyboard/menu inputs |

1. **Creational Pattern**

*Factory Method*

The Factory Method Pattern provides an interface for object creation in a superclass while allowing subclasses to alter the type of objects that are instantiated.

**Implementation in JabberPoint:**

* This pattern should be applied to the SlideItem class to standardize the creation of different types of slide elements (e.g., TextItem, BitmapItem).
* Implementing a factory will separate construction logic from its usage, making it easier to add new slide components in the future.

**Why?**

* In the current code, SlideItem objects are created within various methods that have other responsibilities.
* By introducing a factory, object creation will be centralized, improving code organization and adhering to the Single Responsibility Principle (SRP).

**Added Value:**

* New SlideItem types can be added without modifying existing logic.
* Encapsulates object creation, making the system easier to modify and test.

1. **Structural Pattern**

*Composite*

The Composite Pattern allows treating individual objects and groups of objects uniformly, making it useful for hierarchical structures.

**Implementation in JabberPoint:**

* This pattern should be applied to the Slide class.
* A Slide contains multiple elements (SlideItem objects), which can be text, images, or other nested components.
* Instead of handling each item separately, the Composite Pattern will allow all SlideItem elements to be treated the same way.

**Why?**

* Simplifies slide management by treating text, images, and other components uniformly.
* Enhances scalability by allowing new types of SlideItem elements without modifying the Slide class.
* Makes it easier to manipulate, render, and update slide elements dynamically.

**Added Value:**

* SlideItem elements can be nested or grouped easily.
* Reduces the complexity of managing different slide elements.

1. **Behavioral Pattern**

*Command*

The Command Pattern encapsulates user actions as objects, decoupling request senders from request receivers.

**Implementation in JabberPoint:**

* The KeyController and MenuController should use the Command Pattern to handle:
  + Keyboard shortcuts (e.g., Next Slide, Previous Slide).
  + Menu selections (e.g., Open, Save, Exit).
* Each command (e.g., NextSlideCommand, PrevSlideCommand) encapsulates an action.

**Why?**

* Decouples UI input from execution logic, making it easier to modify and extend.
* Encapsulates event handling, improving code organization.
* Enables undo/redo functionality, as commands can be stored and reversed.

**Added Value:**

* New commands can be added without modifying controllers.
* Command objects can be used across different parts of the application.
* Commands can be unit tested independently.