OOD REPORT

Precious Oziwo

4848888

1 Introduction

Jabberpoint, is a popular presentation software, that has been frequently utilized by professionals and students for years. Nonetheless, Jabberpoint has its share of architectural issues that might affect its usability and performance, just like any complicated piece of software. I will list these issues and suggest solutions for each of them in this report.

The report will outline the issues with the Jabberpoint architecture as it stands right now. These problems could involve, among other things, sluggish performance, poor scalability, and lack of flexibility. Following the identification of these issues, a solution will be offered for each of the problems discovered. These solutions can entail reworking a portion of the software's codebase.

The final step is that each suggested solution will be supported by a convincing justification for why it is a good one. This entails weighing the benefits and downsides of each proposal and illustrating how it will solve the issue at hand. The objective is to offer practical suggestions that will aid Jabberpoint in resolving its architectural issues and enhancing its usability and functionality.

2 Architectural Problems

The architectural issue of JabberPoint is a violation of the single responsibility principle (SRP). The single responsibility principle (SRP) states that a class should have one, and only one, reason to change. Janssen (2023)

1. The MenuController class: This class extends MenuBar and provides information about how various menu items are implemented, making it accountable for a number of functions like controlling the presentation, managing file actions, and displaying the help menu. As a result, the class has an excessive number of responsibilities, which makes it challenging to extend, maintain, and test.

   Additionally, the class also has hard-coded strings, which can make it difficult to change the menu items in the future.

2. XMLAccessor: This class is in charge of loading the presentation data, saving it to an XML file, and parsing the resulting XML file to create a Presentation. The problem here is that both methods are placed in the same class which make It difficult to understand and maintain code. This class also has too many hard-coded strings.

3. Presentation: The initial problem here is the lack of clear encapsulation and the lack of a proper interface for the presentation properties.

4. Style: A static array called styles in the original Style class is used to store instances of the Style class. Although this works, it creates a globally mutable state, which could cause problems in a multi-threaded or concurrent context. Furthermore, it could be argued to be less tidy in terms of concern separation and encapsulation.

2.1 Proposed Solution

Solution 1: The menu controller class should be broken down into multiple classes so that each class will have its own functionality. The new classes derived from the menu controller class are below:

FileMenu class: This class will contain all functionalities for the file menu buttons such as to open, new and save presentation and exit function to close the app.

ViewMenu class: This class will consist of the functionalities to navigate the presentation slide, such as next and previous slide.

HelpMenu class: This class will contain the functionality to navigate to the about page

MakeMenuItem class: The class will have a method that creates the menu item such as File, view and help.

MenuException: This class will handle error messages.

Solution 2: Xml Accessor class: This class should also be broken down into separate classes and 1 interface. The Xml Accessor class will still contain function to load slide items.

SaveXmlFile Class: This class will contain the method for saving the presentation.

LoadXmlFile class: The class contains the method to open a presentation.

LoadAndSaveXml Interface: This interface will contain 2 methods to loadFile and SaveFile

XmlException: This class handles error messages

XmlTag: This is an enum which will be representing a fixed set of constants.

In other for this solution to work, the Accessor class will implement LoadAndSaveXml

Solution 3: This presentation class will be broken down into an interface and other classes.

An interface PresentationSlideProps will be added to enforce a clear contract for classes that need presentation properties.

PresentationFrame class to hold the presentation and the frame, improving code organization.

StartPresentation class that will have 3 methods to open, save and create new presentation.

Solution 4: The solution here involves creating a separate StyleHelperClass to manage the styles. The styles array is encapsulated in this class, which also offers static methods for generating and obtaining styles.

2.2 Argument

The proposed solution is a good one since all methods in both Menu controller and and Xlm Accessor are added to a separate class, it is now easier to understand and maintainable the code. It is also easier to modify the different menu items without modifying the source code. Each of this class is also easier to test.

The proposed solution will also make the application more efficient than the original one because it eliminates unnecessary steps and optimizes the algorithms. The solution also also introduces better separation of concerns and encapsulation, making the classes more modular and adhering to good object-oriented design principles.

3 Reference

Janssen, T. (2023, March 16). *SOLID Design Principles Explained: The Single Responsibility Principle*. Stackify. https://stackify.com/solid-design-principles/