Potential Improvements to Jabberpoint – Advice

This document presents a summary of found issues with the design and code of Jabberpoint, as well as our proposal to refactor it.

Before we can get to refactoring the code, let us first start with the biggest problems we are trying to address with the refactor process.

1. Issues with the current system

1) The system remains tightly coupled, especially around the Presentation class. Multiple objects store a reference to the Presentation to perform functionality like “go to next slide” or “save file”, while the Presentation must directly update the UI anytime it changes. This issue could be resolved with two patterns – Command and Observer, which would both allow the classes to interact with one another without being chained by implementation details.

2) The XMLAccessor class violates the Single Responsibility Principle in several ways. First of all, it has the task of both importing and exporting XML, which are very different operations. Moreover, it also has the job of constructing and populating the Presentation with slide items when it opens a file.

3) Various constants are used as default file names or for xml parsing purposes, but all the constants are scattered across classes. Creating a namespace or (class with classes) to hold constants will allow us to easily access them without having to remember which constant belongs to which class.

2. Design patterns

In order to address these issues with Jabberpoint program, we advise to implement the following design patterns:

2.1 Observer:

As of this moment, any time the user goes to a new slide, the Presentation calls the SlideViewerComponent.update() method to repaint the whole screen. While this does solve the problem of ensuring that the image the user sees on their screen always matches the internal state of the underlying Presentation, it leads to tight coupling between these two classes, and it makes it difficult to update another class based on the Presentation state changes.

The Observer pattern allows us to have a few generic Subscribers/Listeners follow the Publisher class (in our case, Presentation) to subscribe to different event types and process them as they happen, without being constrained by the specific implementation details of the Publisher and the Listener.

While in our case, there is only one Listener (the SlideViewerComponent), the benefits of using an Observer might seem weak compared to the additional complexity it introduces to the code. However, by reducing the number of links between classes, the code will become easier to maintain, and it will be possible to add new features that rely on frequent updates from Presentation, for example a code to autosave a file if the presentation is closed could also be implemented via Observers.

2.2 Command – handle new files, saves, opens, next slide, prev, goto independently of the caller

The Command pattern can be utilised in Jabberpoint to further decouple Presentation from various UI actions triggered by Menu and Key Controllers. Not only do some of the Controllers have shared functionality, which can be instead moved to separate commands to avoid code duplication in those and allow multiple triggers for one action.

While this does have the disadvantage of needing separate classes for each command, this addition to the complexity should be compensated by a simpler and more modular program structure. Additionally, the Command pattern allows us to store the history of user actions, potentially allowing to implement undo/redo functionality, though now, there is little reason to implement this feature, as Jabberpoint does not have any support for editing slide contents.

2.3 Factory – create SlideItems on demand

As yet another improvement, we should move the creation of SlideItems out of XML parser and into the Factory class, which would add the advantage of allowing us to easily implement new SlideItems in the future. Currently, there are only two types of Slide Items – the Bitmap and Text, but should a future developer desire to implement a video or a link item, they would no longer have to change the parsing logic to support them.

Of course, if there are just two classes in one factory, the benefits of using factory method may not be very clear, it reduces the importance of the bloated XMLAccessor class.

Miscellaneous improvements

Split XMLAccessor functionality into XMLLoader and XMLWriter – since read and write operations are noticeably different in implementations, we could have XMLAccessor be an aggregate of Loader and Writer classes, which would be responsible for the implementation of import and export features.

Designated classes to hold constants – having a single Constants class with inner classes to hold constant fields allows us to have a centralised access point to all constants – so if the programmer were to add a new feature to jabberpoint, and that would require a constant, they would not need to think which class to put it in.