Refactor Proposal

JabberPoint

|  |  |
| --- | --- |
| Author: | Ramon Gonzalez Ordaz |
| Student Nr. | 4625544 |
| Date: | 24/04/2023 |
| Class: | IN-OO1 |
| Lecturer: | Arvid Fens |

# JabberPoint

According to the brief documentation on the software itself, Jabber is a crude implementation for a slide presentation software. This brief report proposes a new refactored solution as well as an explanation on the design choices for said solution.

## Findings

The Appendix: Current Implementation Class Diagram, A diagram illustrates the current state of the software.

Beginning with the XMLAccessor and DemoPresentation classes, it can be concluded that they violate the Single Responsibility SOLID principle. This is done when these two classes assume the resposibility of ‘creating’ a Presentation instance (Ecker, 2015). The XMLAccessor also uses a series of temporary fields that are unused. Additionally, the Accessor class returns a sub-class object which creates an uneccessary dependency with a code smell of divergent change (Shvets, Divergent Change).

The Style class is used and created by several classes including the main JabberPoint class. This creates ‘Feature Envy’, this code smell pertains to the constant use of another class’ methods (Shvets, Feature Envy). Moreover, the Style class contains a CreateStyles method that creates ‘hard coded’ styles.

The MenuController also experiences some bloat due to temporary fields and data clumps (Shvets, Temporary Field).

Furthermore, methods in the XMLAccessor and MenuController contain significantly lage methods that exceed 10 lines. A solution to this proble would be to extract the processes form these methods and create new ones that are easier to debug.

## Improvements

**XMLAccessor**

A proposed solution would extract the action of creating a presentation and rather return a presentation object that can be consumed by a different class. The protected methods in the XMLAccessor class can be put in the lower lever class called Accessor. Both

To reduce temporary fields that are used only once, Enumerable values can be introduced.

Furthermore, the Accessor class can be cleaned up by extracting the get method and placing it in the proper sub-class.

**Styles: Feature Envy**

A Style belongs predominantly to a Slide and SlideItem classes. The style classes can be customized when instantiating an object for a dynamic application.

**MenuController**

The class can be extracted from to implement the command pattern. Once again, the use of temporary fields can be removed to introduce Enumerable values or sub-classes with specific behavior. For actions like button-clicks, a Button Abstract class can be introduced to handle actions.

Moreover, the constructor is bloated with initialization code for action listeners. The solution is to apply the Button/Action sub-classes and initializing them in their own methods.

**Final Strings Attributes**

The use of a **final strings** varies throughout the entire application. Some of the classes use it as data variables or as exception messages. These strings attributes can be removed to implement proper exceptions for error handling.

**Slide Items**

Lastly, the Slide and SlideItem classes. The Slide class is slightly bloated with getters and setters that can be moved to the Parent SlideItem abstract class. Only methods that should remain are the inhereted overrideable methods.

# References

Ecker, R. (2015, December 19). *Solid Principles.* Retrieved from The Team Coder: https://team-coder.com/solid-principles/

Shvets, A. (n.d.). *Code Smells.* Retrieved from Refactoring Guru: https://refactoring.guru/refactoring/smells

Shvets, A. (n.d.). *Divergent Change.* Retrieved from Refactoring Guru: https://refactoring.guru/smells/divergent-change

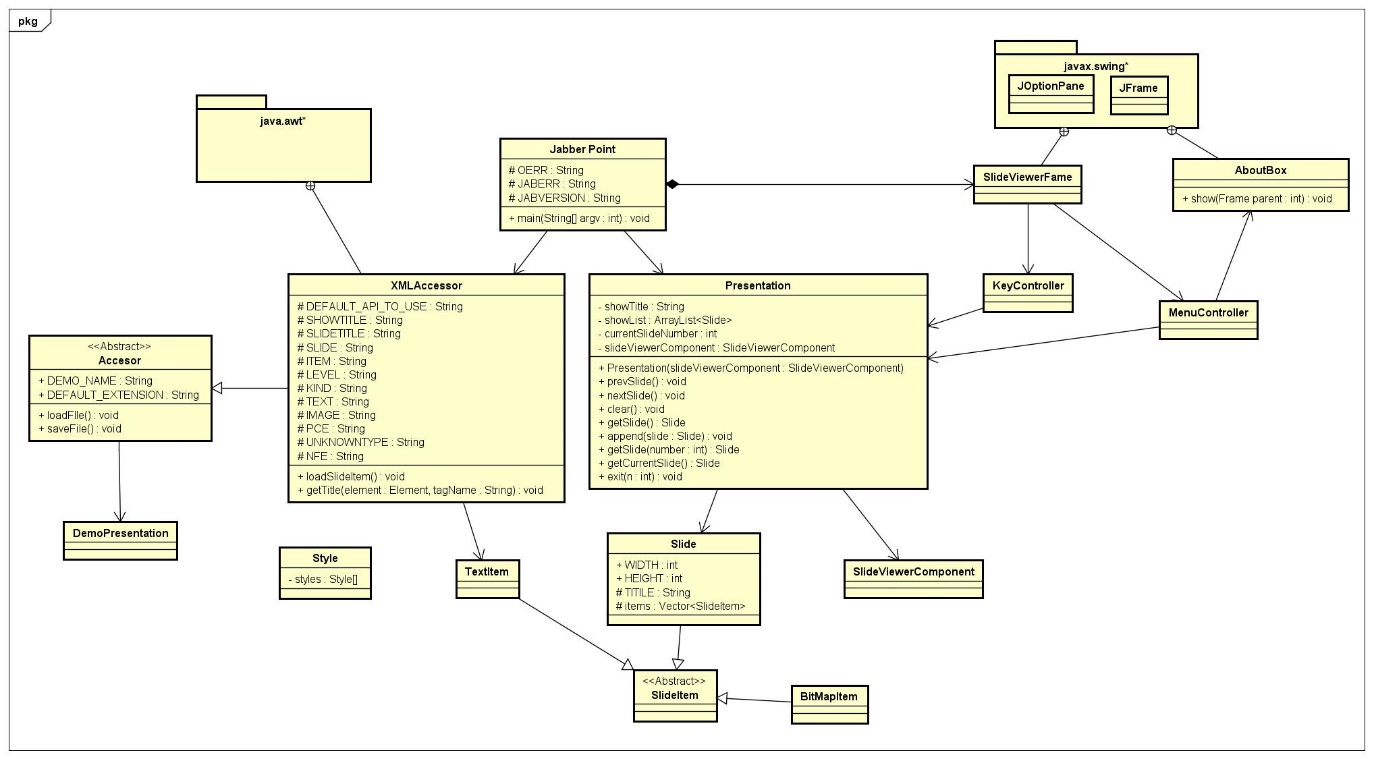
Shvets, A. (n.d.). *Feature Envy.* Retrieved from Refactoring Guru: https://refactoring.guru/smells/feature-envy

Shvets, A. (n.d.). *Switch Statements.* Retrieved from Refactoring Guru: https://refactoring.guru/smells/switch-statements

Shvets, A. (n.d.). *Temporary Field.* Retrieved from Refactoring Guru: https://refactoring.guru/smells/temporary-field

Shvetz, A. (n.d.). *Command.* Retrieved from Refactor Guru: https://refactoring.guru/design-patterns/command

# Appendix: Current Implementation Class Diagram



# Appendix: Refactored diagram.

A picture containing text, diagram, plan, handwriting

Description automatically generated