Issues of the architectural kind within Jabberpoint

Roy Stobbe – 33 60 26 0

07-04-2024

# Overview

The aim of this document is to discuss on a class by class basis which problems have been discovered in the Jabberpoint program and which solutions have been thought of. Certain text has been highlighted in **bold** as to indicate that this contributes to solving or contributing to key parts of the program.

## Unordered classes

Create packages for classes, increasing **Maintainability**

# Accessor

## Accessor is a class with only two abstract methods.

Convert the abstract class into an interface.

## Accessor interface is not suitable because not every implementation will need to load and save.

Create two interfaces from the current singular interface, LoadAble and SaveAble. This allows an implementation to perform either one of the two when necessary. This increases the **adaptability** of the application.

# Style

Style has too many responsibilities.

Allow the creation of Styles to occur in a StyleFactory based on a StyleLevel Enum so that there is less room for errors when creating a Style Object. When a new Style is needed, one can easily create a new enum and adjust the createStyle method accordingly. All of this increases **adaptability** and **maintainability**.

Fields in Style have an insecure access modifier.

Set the fields in Style private and add getters and setters.

When a Style is created with an enum without definition in the factory, this is not clearly communicated to the user.

Create a custom exception that is thrown when a new enum is applied that does not yet have a definition in the code. This increases **testability** and **maintainability**.

The fields in Style are not optimal.

Make the font field final since it does not change. This increases **maintainability**.

Remove the fontsize field as it can be obtained from the font.

# XMLAccessor and DemoPresentation

## Classes instantiate new instances of these classes while it is not their responsibility.

Create a factory for XMLAccessor and DemoPresentation.

## DemoPresentation has a too large method.

Divide the loadFile method into smaller methods to make it easier to test and read. This increases **testability** and **maintainability**.

# AboutBox

## AboutBox is not abstract even though it never needs to be instantiated.

Declare AboutBox as abstract.

# KeyController and MenuController

## Classes instantiate new instances of these classes while it is not their responsibility.

Create a factory for KeyController and MenuController.

MenuController has too large a constructor.

Divide the constructor into separate methods. This makes it easier to test and read. This increases **testability** and **maintainability**.

## MenuController loads and saves files using an Accessor of datatype XMLAccessor.

Provide the accessors with datatypes SaveAble and LoadAble so that in the future, other Accessors can also be used. This increases **adaptability**.

## MenuController sets the slideNumber to 0 independently of loading a file, while these two actions always occur together.

Move the responsibility of setting to the loadFile method so that it always occurs together. This increases **reliability** and **usability**.

# Presentation and SlideViewerComponent

## There is a circular relationship between these two classes.

Arrange it so that a SlideViewerComponent has and controls a Presentation, and not vice versa. Move the methods within Presentation that deal with slides to SlideViewerComponent. All other classes that contain a presentation will instead have a SlideViewerComponent.

## SlideViewerComponent updates the frame every time a slide changes, even though this has no impact on the frame.

Divide and move the update method in SlideViewerComponent to the various classes that are updated, so they can be updated individually. This improves **performance**.