Architecturale problemen binnen Jabberpoint

Thijs Janse 4607023

02-04-2023

# Algemeen

In dit verslag wordt er per class aangegeven welke problemen er gevonden zijn en welke oplossing daarvoor bedacht is. Mocht deze oplossing bijdragen aan 1 van de key onderdelen ven een goed ontwerp, dan is dit dikgedrukt aangegeven.

## De classes zijn ongeordend

Packages aanmaken voor de classes. Dit verhoogt de **Maintainability**

# Accessor

## Accessor is een class met enkel twee abstracte methodes.

Van de abstracte class een interface maken

## Accessor interface is niet goed toe te passen omdat niet elke implementatie zal moeten loaden en saven

Van de interface twee interfaces maken, LoadAble en SaveAble. Zo kan een implementatie ook 1 van de twee doen indien nodig. Dit vergroot de **Adaptability** van de applicatie.

# Style

## Style heeft te veel verantwoordelijkheden

Het aanmaken van Styles laten gebeuren in een StyleFactory aan de hand van een StyleLevel Enum zodat er bij het maken van een Style Object minder vrijheid is fouten te maken. Wanneer een nieuwe Style nodig is kan men gemakkelijk een nieuwe enum maken en de createStyle naar toebehoren aanpassen. Dit alles vergroot de **Adaptability** en **Maintainability**

## Fields in Style hebben een onveilige accessmodifier

De fields in Style private maken en getters en setters toevoegen

## Wanneer een Style wordt aangemaakt met een enum zonder invulling in de factory wordt dit niet duidelijk gemaakt aan de gebruiker

Een custom exception maken die wordt gegooid wanneer er een nieuwe enum wordt toegepast die nog geen invulling in de code heeft. Dit vergroot de **Testability** en **Maintainability**

## De fields in Style zijn niet optimaal

Field font final maken omdat deze niet meer verandert. Dit verhoogt de **Maintainability.**

Field fontsize verwijderen omdat deze uit font te halen is

# XMLAccessor en DemoPresentation

## Classes maken nieuwe instanties van deze classes aan terwijl dit niet hun verantwoordelijkheid is

Een factory maken voor XMLAccessor en DemoPresentation.

## DemoPresentation heeft een te grote methode

De methode loadFile opdelen in kleinere methodes zodat het makkelijker te testen en lezen is. Dit vergroot de **Testability** en **Maintainability**

# AboutBox

## AboutBox is niet abstract terwijl er nooit een instantie van gemaakt hoeft te worden

AboutBox abstract verklaren

# KeyController en MenuController

## Classes maken nieuwe instanties van deze classes aan terwijl dit niet hun verantwoordelijkheid is

Een factory maken voor KeyController en MenuController.

## MenuController heeft een grote constructor

Constructor opdelen in losse methodes. Dit maakt het makkelijker te testen en te lezen. Dit verhoogt de **Testability** en **Maintainability**

## MenuController laadt en slaat files op met een Accessor van datatype XMLAccessor

De accessors de datatypes SaveAble en LoadAble geven zodat dit in de toekomst ook andere Accessors kunnen zijn. dit vergroot de **Adaptability**

## MenuController set de slideNumber naar 0 los van het laden van een file. Terwijl deze 2 dingen altijd samen gebeuren

De verantwoordelijkheid van het het setten naar de loadFile methode verplaatsen zodat het altijd samen gebeurt. Dit verhoogt de **Reliability and Usability**

# Presentation en SlideViewerComponent

## Er is een circelvormige relatie tussen deze twee classes

Het zo maken dat een SlideViewerComponent een Presentation heeft en bestuurd, en niet andersom. Door de methodes binnen presentation die over slides gaan te verplaatsen naar de SlideViewerComponent. Alle andere classes die een presentation containen zullen in plaats daarvan een SlideViewerComponent hebben.

## SlideViewerComponent update het frame elke keer als er een slide verandert terwijl dit geen invloed op de het frame heeft.

De update methode in SlideViewerComponent opdelen en verplaatsen naar de verschillende classes die ge-update worden. Zodat deze individueel ge-update kunnen worden. Dit verbetert de **Performance**