Refactoring Jabberpoint

Object Geörienteerd ontwerpen

Monique Sabong - 5000742

2023

Inhoud

[**Inleiding** 2](#_Toc131348087)

[Bug fix “go to” 3](#_Toc131348088)

[Utility 3](#_Toc131348089)

[MenuController 3](#_Toc131348090)

[Accessor 3](#_Toc131348091)

[XML Accessor 4](#_Toc131348092)

[Kleine aanpassingen 4](#_Toc131348093)

[Style 4](#_Toc131348094)

# **Inleiding**

Voor het vak Object Georiënteerd Ontwerpen moet ik het programma Jabberpoint refactoren. Jabberpoint is een presentatie programma die vreemd is opgebouwd. Aan mij de taak om Jabberpoint hetzelfde te laten te functioneren, maar beter op te bouwen. In dit document ga ik langs de veranderingen die ik in Jabberpoint heb aangemaakt.

De volgende punten zijn meegeleverd in mijn GitHub:

* Dit document;
* Use-case diagram;
* Activity diagrams;
* Een geschatte klassendiagram van Jabberpoint;
* De klassendiagram van Jabberpoint voor het refactoren;
* De klassendiagram van Jabberpoint na refactoring.
* De code van Jabberpoint na refactoring

# Bug fix “go to”

Wanneer je de programma opstart en in de menu op “View” en daarna op “Go to” klikt, kan je een nummer intikken die geen slide heeft. Hierbij gaat alleen de slide counter omhoog.

Om dit op te lossen heb ik in de methode “setSlideNumber(int number)” een check toegevoegd, die kijkt of de slide number bestaat. Als dat niet het geval is, blijft de huidige slide staan en blijft de counter op de huidige slide staan. Nu gaat de counter niet meer naar een hoge niet bestaande getal.

# Utility

In verschillende klassen heb je “public static final” fields staan. Het nadeel hiervan is als je iets wilt aanpassen dat het wel zoekwerk kan zijn, wat tijd kan kosten.

Om dit op te lossen heb ik een utility package aangemaakt met meerdere klassen, zoals error messages, file messages, menu items en slide frame. Hierin staan alle static finals in dus als je in de toekomst iets wilt aanpassen kan je altijd naar de Utility package gaan. Alles staat nu in één centrale plek en hoef je nu niet meer te zoeken wat tijd kan schelen.

# MenuController

In de klasse MenuController gebeuren twee dingen. De menu controller maakt de menu items aan voor in het menu bar en hij creëert de acties van de menu items. Dat betekent dat de klasse twee verantwoordelijkheden heeft.

Gebruikte bron: <https://refactoring.guru/design-patterns/factory-method>

Om dit op te lossen heb ik een nieuwe klasse aangemaakt genaamd “MenuItemFactory” die de taak, het maken van menu items, overneemt van de MenuController klasse. Nu is de MenuController klasse alleen verantwoordelijk voor de acties van de menu items.

# Accessor

In de klasse Accessor gebeuren er twee dingen: het lezen/laden en het opslaan van presentaties. De XMLAccessor en de DemoPresentation klassen worden beiden uitgebreid door de Accessor klasse, waarbij de DemoPresentation klasse alleen maar de load functie gebruikt en niet de save functie.

Om dit op te lossen heb ik de klasse Accessor in twee interfaces gesplit “Readable” en “Writable”. De klasse “Readable” neemt de loadFile() functie over en en de klasse “Writable” neemt de functie saveFile() over. Hierdoor kan de DemoPresentation alleen de loadFunctie extenden en in de toekomst kunnen klassen worden uitgebreid met alleen de benodigde functie(s).

# XML Accessor

In de klasse XML Accessor gebeuren er twee dingen: laden en opslaan. Hierbij heeft deze klasse twee verantwoordelijkheden.

(Single Responsibility Principle)

Om dit op te lossen heb ik de XMLAccessor klasse in twee andere klassen gesplits: XMLReader en XMLWriter. Hierbij krijgt XMLReader alle load functies en de XMLWriter de save functie. Hierdoor is de code makkelijker te onderhouden en uit te breiden in de toekomst. Ook kan de XMLReader en XMLWriter klasse hergebruikt worden.

# Kleine aanpassingen

* this. Voor variabelen geplaats.
* In de klasse SlideItem zit een tweede lege contructor die niet werd gebruikt, die heb ik weggehaald.
* In de klasse TextItem zit ook een tweede constructor die niet gebruikt werd, die heb ik ook weggehaald.
* Hetzelfde geld voor de klasse BitmapItem, daar heb ik ook een niet gebruikte constructor weggehaald.

# Style

In de klasse Style worden Style objecten aangemaakt en wordt er beschreven wat een style moet hebben.

Ik heb een Style Factory aangemaakt dit de verantwoordelijkheid overneemt om style objecten te creëren.