Object Georiënteerd Ontwerpen

Stefan Meijer, 5033454, Student Informatica NHL Stenden

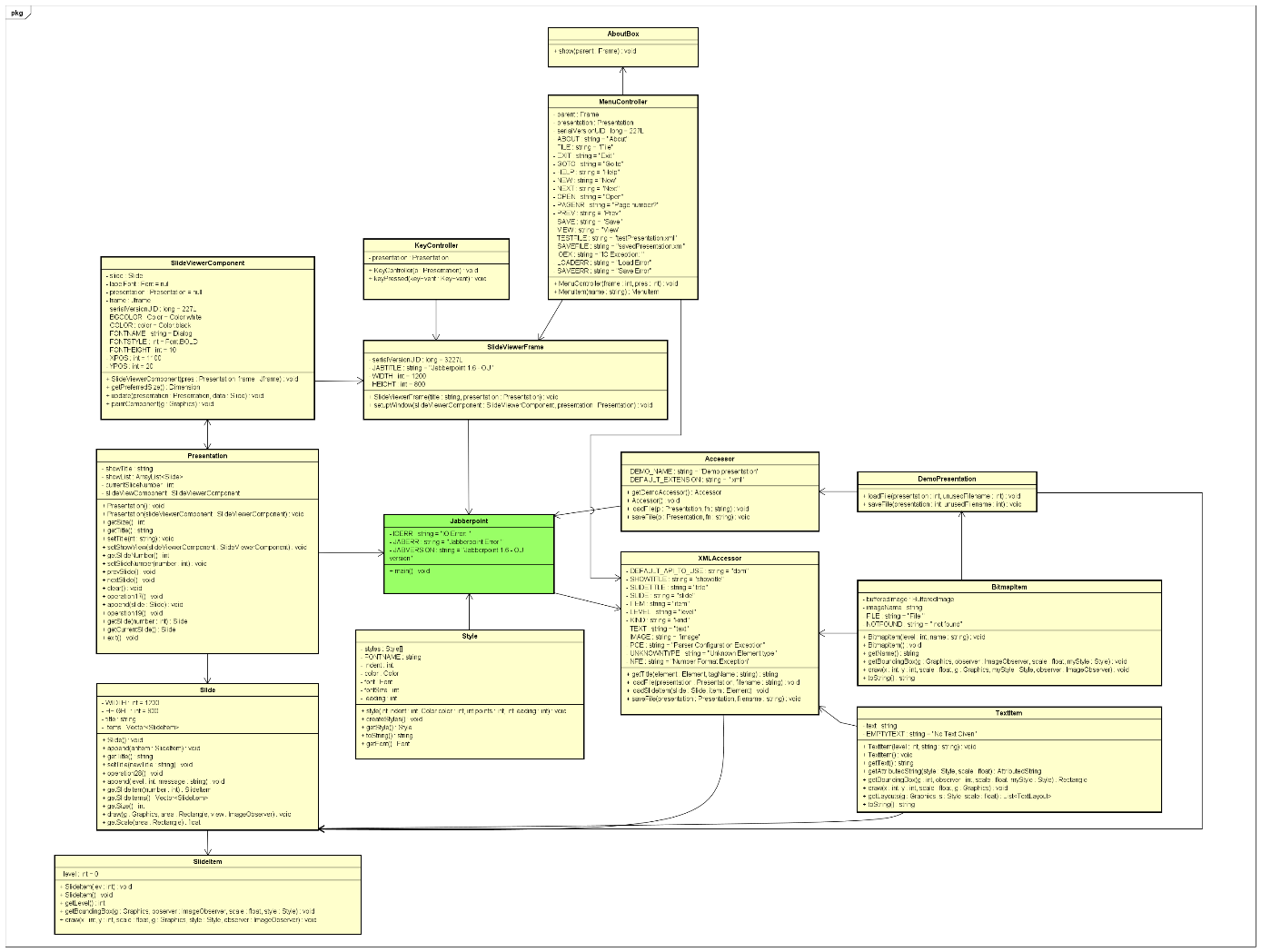
# Inleiding

Jabberpoint is een presentatietool, waarin XML-presentaties worden geladen. Deze applicatie is gemaakt met fouten in de applicatie, waardoor bepaalde functionaliteiten niet goed zijn opgebouwd of niet functioneren naar behoren. Het doel van dit document is, om alle verbeteringen aan Jabberpoint te bespreken.

Als allereerst wordt er een klassendiagram getoond van hoe de klassen in Jabberpoint met elkaar communiceren. Daarna worden de verbeteringen besproken. Tot slot wordt er een klassendiagram getoond van de verbeterde variant van de applicatie Jabberpoint.

# Voorafgaand klassendiagram

Om een goed inzicht te krijgen in de applicatie Jabberpoint en de gebruikte klassen van de applicatie, wordt er een voorafgaand klassendiagram opgesteld. Voor het voorafgaande klassendiagram, zie figuur 1.



Figuur 1 - Voorafgaand klassendiagram

# Verbeteren code, zonder klassen te verwijderen/toevoegen/opsplitsen

Bij het maken van het voorafgaande klassendiagram, merkte ik dat er in Jabberpoint code zit dat niet wordt gebruikt(overbodige code), variabelen/methode die op de verkeerde zichtbaarheid staan en verkeerde casting.

* Ik heb op alle klassen getters & Setters toegepast.
* Alle waarden die direct toegewezen werden aan een variabele en die niet static final waren, heb ik naar de constructor verplaatst.
* Ik heb de zichtbaarheid goed gezet voor variabelen/methoden.
* Ik heb ongebruikte code verwijderd.
* Code logischer neergezet (constructor bovenaan, daarna de methoden en dan de getters&setters). Dit zorgt voor meer overzicht.
* Op advies van Intelij heb ik code, wat op een slimmere manier geschreven kan worden, automatisch laten refactoren.

# Folder structuur

Momenteel staan alle assets in de hoofdmap, verspreid en tussen elkaar door. Hiervoor heb ik een map aangemaakt “Assets” en hierin heb ik alle (externe) bestanden verwerkt die worden gebruikt door de applicatie.

* Assets
  + Images
    - .Gif en .jpg
  + Presentations
    - .Dtd en .xml

Daarna heb ik de source map opnieuw ingedeeld. Alle klassen staan verspreid en tussen elkaar door, dit maakt het erg onoverzichtelijk om mee te werken. Met deze nieuwe structuur kan er rekening gehouden worden met eventuele factory’s en het opsplitsen van klassen.

* Src
  + Accessor
    - Accesor.java
    - DemoPresentation.java
    - XMLAccessor.java
  + Presentation
    - Presentation.java
  + Slide
    - BitmapItem.java
    - Slide.java
    - SlideItem.java
    - SlideViewerComponent.java
    - SlideViewerFrame.java
    - TextItem.java
  + Style
    - Style.java
  + Utility
    - AboutBox.java
    - KeyController.java
    - MenuController.java
  + JabberPoint.java

Deze structuur wordt veranderd in de loop van dit document bij het aanpassen/toevoegen van klassen.

# StyleFactory klasse

In de code worden momenteel op allerlei plekken klassen aangemaakt, waar dit niet hoort. Hiervoor heb ik factory’s aangemaakt. Factory Method is een creatieontwerppatroon dat een interface biedt voor het maken van objecten in een superklasse, maar waarmee subklassen het type objecten kunnen wijzigen dat wordt gemaakt (Refactoring.Guru, z.d.-a).

Momenteel worden er in de style klasses Styles aangemaakt. Dit is natuurlijk niet de bedoeling. Hiervoor heb ik een factory Aangemaakt. Hierin staan nu alle styles die werden aangemaakt binnen de styles klasse in een nette Array. De aangepaste structuur voor de package Style ziet er nu zo uit:

* Style
  + Style.java
  + StyleFactory.java

# SlideFactory klasse en Slide package

Er worden momenteel meerdere slides aangemaakt in Accessor klassen. Hiervoor heb ik een SlideFactory opgesteld. Ook heb ik de klassen in de package Slide anders gestructureerd.

* Slide
  + Item
    - BitmapItem.java
    - SlideItem.java
    - TextItem.java
  + Viewer
    - SlideViewerComponent
    - SlideViewerFrame
  + Slide.java
  + SlideFactory.java

# SlideItemFactory

In de klasse Demopresentation worden een boel Slides.append() met hierin TextItems. Momenteel gaat dit heel omslachtig, eerst wordt een functie in slide aangeroepen, die vervolgens dezelfde functie, maar met andere parameters aanroept, die daarin een TextItem aanmaakt.

Dit zelfde geldt voor BitmapItem. Hiervoor heb ik een SlideItemFactory aangemaakt, omdat TextItem en SlideItem extenden. SlideItem zelf is een abstracte klasse, TekstItem en BitmapItem niet. Ik heb de factory ook toegepast op andere klassen die gebruik maken van SlideItems.

De klassen in de package Item onder Slide zijn nu zo gestructureerd:

* Slide
  + Item
    - BitmapItem.java
    - SlideItem.java
    - SlideItemFactory.java
    - TextItem.java

# Menucontroller

Terwijl ik deze klasse aan het bekijken was, zag ik dat er een heleboel functies werden gemaakt in de constructor (Bloater), Bloators zijn codes, methoden en klassen die tot zulke gigantische proporties zijn toegenomen dat het moeilijk is om ermee te werken (Design patterns and refactoring, z.d.). Deze heb ik allemaal in een eigen methode gezet.

## Goto functionaliteit

In de Menucontroller zit de functionaliteit of nu methode gotoPageAction, die er voor zorgt dat het mogelijk is om naar een specifieke slide te navigeren. Bijvoorbeeld go to (ga naar) slide 1/3/5/10 etc. Het probleem hiermee echter is, dat deze functionaliteit het ook accepteert om naar slide 10 te gaan, terwijl de presentatie slechte 5 slides bevat, naast dat het mogelijk is om over het aantal slides te gaan, is het ook mogelijk om een min waarde in te voeren, bijvoorbeeld: -5. Wanneer er een min waarde wordt ingevoerd, gaat de applicatie stuk, hetzelfde geldt voor een tekst waarde (string).

Hiervoor heb ik een try catch blok gemaakt, dat probeert van de string input een integer te maken, als dit niet lukt betekend het dat er een string is geweest als input, vervolgens krijgt de gebruiker een bericht terug dat alleen nummers toegestaan zijn. Mocht er wel een nummer (int) worden ingevoerd en deze valt niet boven 0 of boven het maximum aantal slides, gebeurt er nu niks.

## MenuFactory

Zo werd er in MenuController ook een methode gedefinieerd die een MenuItem aanmaakt en wordt er een MenuController aangemaakt. Hiervoor heb ik een MenuFactory aangemaakt.

## Menu shortcuts

In Jabberpoint kun je gebruik maken van shortcuts om door de applicatie heen te navigeren, echter zit hier een fout in bij het punt, dat wanneer je de shortcut van volgende slide wilt gebruiken. Hiermee wordt een nieuwe presentatie geopend, waarom? Omdat de shortcuts voor een nieuwe presentatie ook op CRTL + N staat, dit komt doordat de eerst letter van het woord als shortcut wordt gebruikt.

Hiervoor heb ik de klasse MenuShortcut bekeken en hierin zag ik dat je een index of useShiftModifier kon aangeven. Ik heb gekozen voor de 2e, de useShiftModifier, omdat er dan geen knoppen veranderd hoeven te worden. Nu heeft een nieuwe presentatie aanmaken de shortcut CTRL + N en naar de volgende slide gaan CRTL + SHIFT + N en zijn de rest van de knoppen dezelfde shortcut gebleven.

Vanwege deze veranderingen heb ik ervoor gekozen om alles wat bij Menu hoort, in een aparte package te plaatsen, Menu en Utility zijn nu zo gestructureerd:

* Utility
  + AboutBox.Java
  + KeyController.java
* Menu
  + MenuController.java
  + MenuFactory.java

# Accessor

Ik zag dat er Accesors werden aangemaakt in menucontroller, waarop ik besloten heb om voor alle Accessor klassen een Factory aan te maken genaamd: AccessorFactory, hierin maak ik een XMLAccessor en DemoPresentation aan.

Terwijl ik de DemoPresentation aan de factory aan het toevoegen was, bedacht ik mij dat deze helemaal niet in een Accessor hoort, het is namelijk zo dat deze wel gebruik maakt van de Accessor klasse, maar deze maakt alleen gebruik van de “loadFile” methode. Naast dat was er dus een methode overgenomen van Accessor “SaveFile” die niet van toepassing is op de demopresentation, maar alleen op XMLAccessor. Hiervoor heb ik Accessor opgesplitst in 2 aparte klassen, namelijk 2 interfaces: LoadController en SaveController.

Daarna heb ik op XMLAccessor de load en save interfaces geïmplementeerd, hiervoor moest de Factory ook aangepast worden, ik heb de demopresentatie hier uit gehaald, omdat het een presentatie is, heb ik deze onder presentatie geplaatst en geinherit en de load interface geimplementeerd.

## PresentationFactory

Omdat Demopresentation nu onderdeel is van Presentation, heb ik een PresentationFactory aangemaakt, die alle presentation aanmaakt waaronder DemoPresentation. Dit houdt in dat de klasse DemoPresentation overbodig is geworden, deze is nu als methode binnen de PresentationFactory opgenomen.

Na al deze veranderingen is er behoorlijk wat geschoven in de packages. De nieuwe lijst voor Accessor, Presentation en Utility is alsvolgt:

* Accessor
  + AccessorFactory.java
  + XMLAccessor.java
* Presentation
  + Presentation.java
  + PresentationFactory.java
* Utility
  + Aboutbox.java
  + KeyController.java
  + LoadController.java
  + SaveController.java

# Static final variabelen

Momenteel worden er op veel plekken vaste waarden gedefinieerd, dit zorgt voor veel onoverzichtelijkheid. Hierom heb ik deze variabelen in een aparte klasse gestopt, waarvan een klasse onoverzichtelijk werd. De klassen die zijn toegevoegd zijn als volgt:

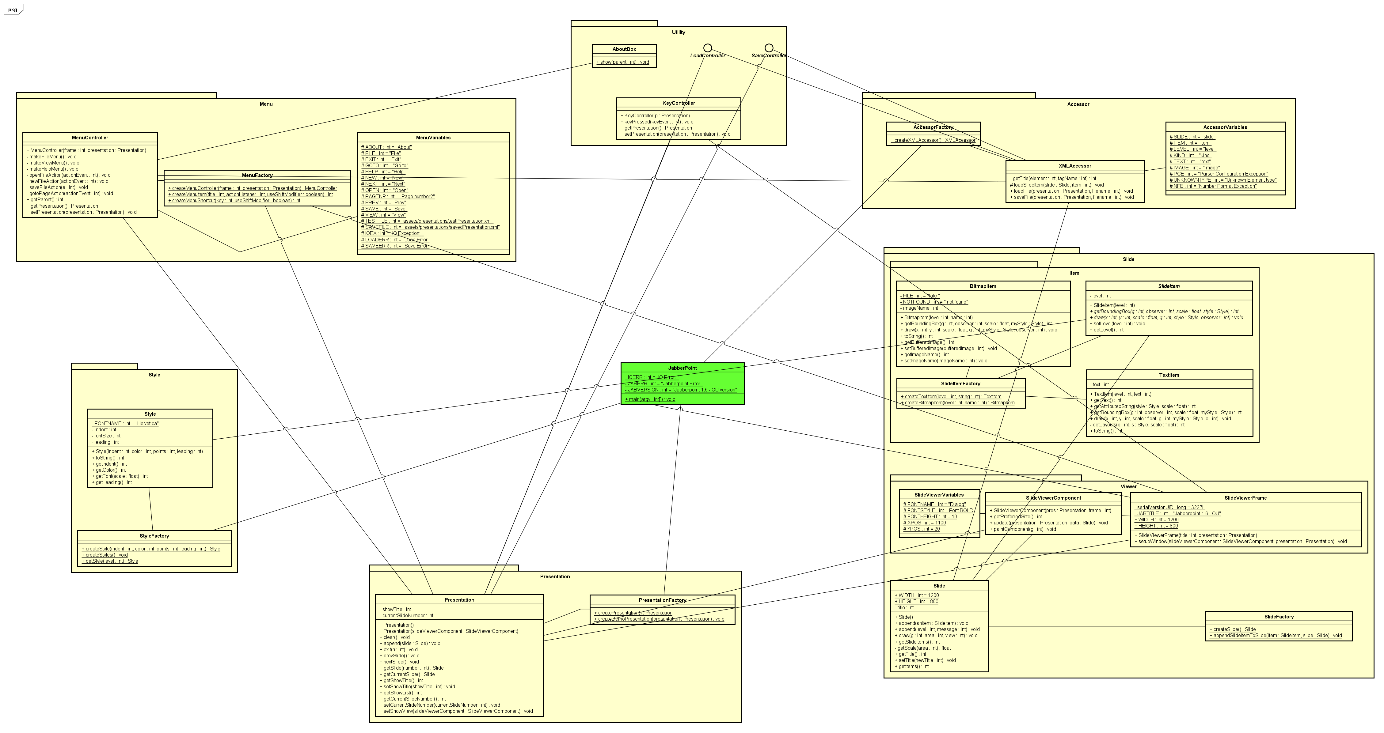
* Accessor
  + AccessorVariables.java
* Menu
  + MenuVariables.java
* Slide
  + Viewer
    - SlideViewerVariables

# Code commentaar toevoegen en nieuwe ongebruikte code verwijderen

Om de code beter leesbaar te maken is er code commentaar toegevoegd, zodat het beter begrijpbaar is, wat er precies gebeurt in de code. Daarnaast is er ongebruikte code verwijderd. Dit is code wat niet werd gebruikt, door geen enkele methode of klasse (denk hierbij aan import, variabelen etc.), dat was overgebleven na het verbeteren van de klassen.

# Definitieve Klassendiagram

Om een overzicht te krijgen van alle klassen die nu Jabberpoint aansturen en hoe deze onderling met elkaar communiceren is er een nieuw klassendiagram opgesteld. Het huidige klassendiagram is te vinden in figuur 2.



Figuur 2- Definitieve Klassendiagram

# Literatuurlijst

1. *Design patterns and refactoring*. (z.d.). <https://sourcemaking.com/refactoring>
2. Refactoring.Guru. (z.d.-b). *Refactoring: clean your code*. https://refactoring.guru/refactoring
3. Refactoring.Guru. (z.d.-a). *Design patterns*. https://refactoring.guru/design-patterns