* Dode code verwijderen?
* Code netjes neerzetten, voor betere leesbaarheid, nu staat alles door elkaarheen
* Getters en setters maken
* Alle niet static/final waarden in constructor defineren
* camelCase toepassen overal

Slideviewercomponent

* Fontstyle is int, maar er wordt font.bold gezet

Demopresentation

* Maakt slide aan in function, gebruik hiervoor factory

Object Georiënteerd Ontwerpen

Stefan Meijer, 5033454

Na het maken van de klassen diagram, ben ik begonnen met het opruimen en het verbeteren van de code.

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

Bij het maken van het 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’s
  + Presentations
    - Dtd en gifs

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).

https://refactoring.guru/design-patterns/factory-method

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

# Presentation heeft SLIDES die kunnen in een aparte klasse

# Bloaters

In de code is veel sprake van Bloaters, 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.).

# 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