# Algemeen

## This

Er ontbreken veel “this.”. Dit zorgt voor onleesbare code en kan soms ook leiden tot het gebruik van de verkeerde variabelen. De oplossing voor dit probleem is voor alle keren dat een lokaal field gebruikt wordt, een “this.” Er voor wordt gezet. Dit zorgt uiteindelijk voor leesbaardere code.

## Openbare variabelen

Er zijn veel openbare variabelen. Dit betekend dat deze variabelen overal ge-set kunnen worden. Deze variabelen mogen alleen gelezen kunnen worden, dus is het beter om ze private te maken en getters toe te voegen. Alle plekken waar deze velden worden opgeroepen zullen dus ook vervangen moeten worden door getters. Dit zorgt er voor dat niet alle variabelen van buitenaf aangepast kunnen worden en zorgt er voor dat waarden niet per ongeluk aangepast kunnen worden.

## Finals

Er ontbreken veel “final” keywords. Hierdoor kan er per ongeluk geschreven worden naar een variabel terwijl het alleen maar uitgelezen wordt. De oplossing voor dit probleem is om voor alle fields die niet veranderen, een “final” voor te zetten. Dit zorgt voor leesbardere code en voorkomt dat de variabelen aangepast worden terwijl dit niet de bedoeling is.

## Nutteloze comments

Er zijn veel nutteloze comments. Zoals bij een field uitleggen waarom het bestaat, of comments bij getters en setters. Deze zijn overbodig. De oplossing voor dit probleem is om alle nutteloze comments te verwijderen. In plaats hier van zal boven elke functie (exclusief getters, setters) javadoc komen. Dit zorgt er voor dat de code makkelijker te begrijpen is.

## Volgorde

Op veel plekken staan functies op een onlogische manier onder elkaar, zoals getters tussen andere functies door, of functies die elkaar overloaden met functies er tussen. De oplossing voor dit probleem is om de volgorde van de functies aan te passen zodat de bij elkaar horende functies onder elkaar staan. Dit zorgt voor beter leesbaardere code.

## Onlogische naamgevingen

Het project bevat veel onduidelijke naamgevingen. Dit kan het lastig maken voor een programmeur om te begrijpen wat een functie of variabele doet. De oplossing voor dit probleem is om de naamgevingen aan te passen zodat de code leesbaarder en makkelijker te begrijpen wordt.

## toString

Er staan in een aantal klassen een toString methode. Deze methoden zijn overbodig en worden nergens gebruikt. Deze kunnen verwijdert worden. Dit zorgt voor beter leesbaardere en overzichtelijkere code.

## Packages

Alle klassen zijn in packages gezet. Voorheen zaten alle klassen in één map. Als alles in packages is onderverdeeld wordt de code beter leesbaar en bruikbaar.

# Accessor

## Accessor opgesplitst

De accessor is verantwoordelijk voor het laden en opslaan van presentaties. DemoPresentation en XMLAccessor extenden beide de Accessor. DemoPresentation gebruikt alleen de load functie en laat de save functie leeg. De accessor kan beter worden opgesplitst in twee klassen, Saver en Loader. De DemoPresentatie extend dan alleen de Loader en XMLAccessor extend dan Saver en Loader. Dit zorgt er voor dat er geen lege functies gecreëerd hoeven te worden.

## getDemoPresentation

deze functie in Accessor wordt opgeroepen wanneer er geen presentatie wordt ingeladen. De functie maakt een nieuwe DemoPresentation aan en returned deze om de demo presentatie in te laden. Deze hele tussenstap kan er uit door de DemoPresentation aan te maken waar de functie wordt opgeroepen. Dit zorgt voor minder onnodige code.

## Veel variabelen

In de klasse XMLAccessor staat een hele hoop variabelen. Deze variabelen geven de namen aan van de XML elementen in de XML bestanden die worden ingeladen. Deze variabelen worden maar één keer gebruikt. Daarnaast gaat de hele XMLAccessor stuk wanneer deze variabelen veranderen, aangezien de namen zijn veranderd waardoor de reader de bestanden niet meer kan uitlezen. Het verwijderen maakt het aanpassen van de XML namen lastiger maar de code wordt er wel overzichtelijker van.

# Styles

## 1. Style array

Styles bevat een array van Styles. Hierdoor krijg je dat je een Style hebt met lege style waarden, met daar binnenin een aalstal Styles met style waarden en met lege arrays.

Dit is op te lossen door een aparte klasse te maken met een array van Styles. Deze klasse zal dan de “StyleFactory” klasse heten. Hierin zal een functie staan waarin op basis van een enum (zie punt 2) een Style object zal worden aangemaakt en gereturned.

Dit zorgt er voor dat er geen lege velden meer zullen ontstaan in de Style klasse.

## 2. Primitive obsession

De style wordt gekozen aan de hand van een level. Dit is een interger. Hierdoor kan het lastig zijn te begrijpen welk getal bij welke style hoort.

Een oplossing voor dit probleem is om de interger om te zetten naar een enum.

Dit zorgt er voor dat je duidelijk kan zien welke style level bij welke style hoort.

## 3. Dode code

Er is een functie genaamd toString die niet gebruikt wordt. Dit is dode code en ook speculative generality.

De oplossing voor dit probleem is om de functie te verwijderen.

Dit zorgt voor beter leesbare en begrijpbare code.

# Presentation

## 1. Slide controls

Het wisselen tussen actieve slides gebeurt in de presentatie klasse. Het zou netter zijn als de pagina controle zijn eigen klasse zou krijgen. Dan blijft de Presentation klasse alleen voor het drawen op het scherm. Dit zorgt voor overzichtelijkere code.

## 2. Slechte pagina checks

Wanneer er naar een pagina genavigeerd wordt, wordt er slecht gecontroleerd of de ingevulde waarde een valide pagina is. Dit kan er voor zorgen dat een gebruiker naar een paginanummer voorbij de slides kan navigeren. Dit probleem is op te lossen door te controleren of de ingevulde waarde tussen de 0 en aantal pagina’s ligt. Dit zorgt er voor dat de gebruiker binnen de slides blijft.

## 3. Exit

Er staat een exit functie in de presentation die beter ergens anders geplaatst kan worden. Deze kan beter verplaatst worden zodat deze functie direct wordt aangeroepen wanneer er op de exit knop wordt gedrukt. Dit zorgt voor een duidelijkere structuur tussen de klassen.

## 4. Ongebruikte functies

Er staan een aantal ongebruikte functies in de presentatie klasse, zoals de tweede constructor. Deze kunnen verwijdert worden om de leesbaarheid van de code te verhogen.

# Slide

## 1. Vectors

Op een aantal locaties worden Vectors gebruikt als collectie. Vectors zijn legacy en zijn over het algemeen ook langzamer dan bijvoorbeeld ArrayLists.

De oplossing voor dit probleem is om alle Vectors te vervangen voor ArrayLists. Er zijn geen speciale features van Vectoren die hier gebruikt, die ArrayLists niet hebben. Ze kunnen dus worden vervangen door een ArrayList.

Dit zorgt uiteindelijk voor nettere, leesbaardere en snellere code.

## 2. Hele objecten

Er wordt aan sommige functies een heel “Presentation” object meegestuurd. Hiervan worden vervolgens alleen een paar waarden gebruikt.

De oplossing voor dit probleem is om alleen de benodigde waarden mee te sturen aan de functie.

Dit verminderd de complexiteit van de code en maakt het makkelijker te lezen.

## 3. Titel

In een SlideViewerFrame wordt een SlideViewerComponent aangemaakt. Vervolgens wordt aan de SlideViewerComponent een SlideViewerFrame object meegestuurd. Deze wordt gebruikt om de titel van de SlideViewerFrame aan te passen. Ook wordt de titel meerdere keren aangepast. Al deze aanpassingen worden alleen overschreven door de titel van de presentatie. Deze aanpassingen zijn dus overbodig.

Deze titel kan beter aangepast worden bij het inladen van een nieuwe presentatie, omdat de titel gezet wordt naar de presentatie titel. De dubbele setTitles kan worden opgelost door de ongebruikte setTitles te verwijderen.

Dit zou voorkomen dat een heel SlideViewerFrame object wordt meegestuurd wat zorgt voor beter begrijpbare code.

## 4. Styles

Er worden in de draw en getBoundingBox functies SlideItems aangemaakt met een style er aan gekoppeld. Vervolgens wordt deze style opgehaald en teruggestuurd naar de SlideItem. Dit is overbodig en zorgt voor verwarring. Dit is op te lossen door de SlideItem zijn eigen style waarde te laten gebruiken. Dit zorgt voor duidelijkere code.

# SlideItem

## 1. Observer

Er wordt een observer meegegeven aan een aantal draw functies. Deze wordt vervolgens alleen gebruikt in de BitmapItem class om een afbeelding te tonen en is hier niet eens verplicht. Deze dode code kan verwijdert worden. Dit zorgt voor nettere code.

## 2. Ongebruikte constructor

Er staat een ongebruikte lege constructor. Deze kan verwijdert worden. Dit zorgt voor nettere code.

# XMLAccessor

## getSlideItems

De accessor vraagt alle slideItems op om vervolgens de lengte van de array en een slideitem er uit te halen. Het zou makkelijker zijn om een functie te maken in Slide die de lengte van de slideItem array geeft en een functie die een slideItem uit de array geeft. Dit zorgt er voor dat er geen tijdelijke variabelen worden aangemaakt, wat zorgt voor leesbaardere code.

# Load/Save static

De XMLAccessor heeft geen eigen vairabelen en vereist ook niet om geïnstantieerd te worden. De functies in de klassen kunnen dan ook statisch worden. Dit zorgt er voor dat de klasse niet elke keer geïnstantieerd moet worden en dat alle klassen overal makkelijk bij de save/load functies kunnen.

# MenuController

## Lange functie

De constructor van de MenuController klasse is heel groot. Deze zal worden opgesplitst in verschillende functies. Dit zorgt voor beter leesbardere code.