# Algemeen

## 1. 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.

## 2. 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.

## 3. 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.

## 4. 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.

## 5. 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.

## 6. 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.

# 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.

# 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.

# 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.