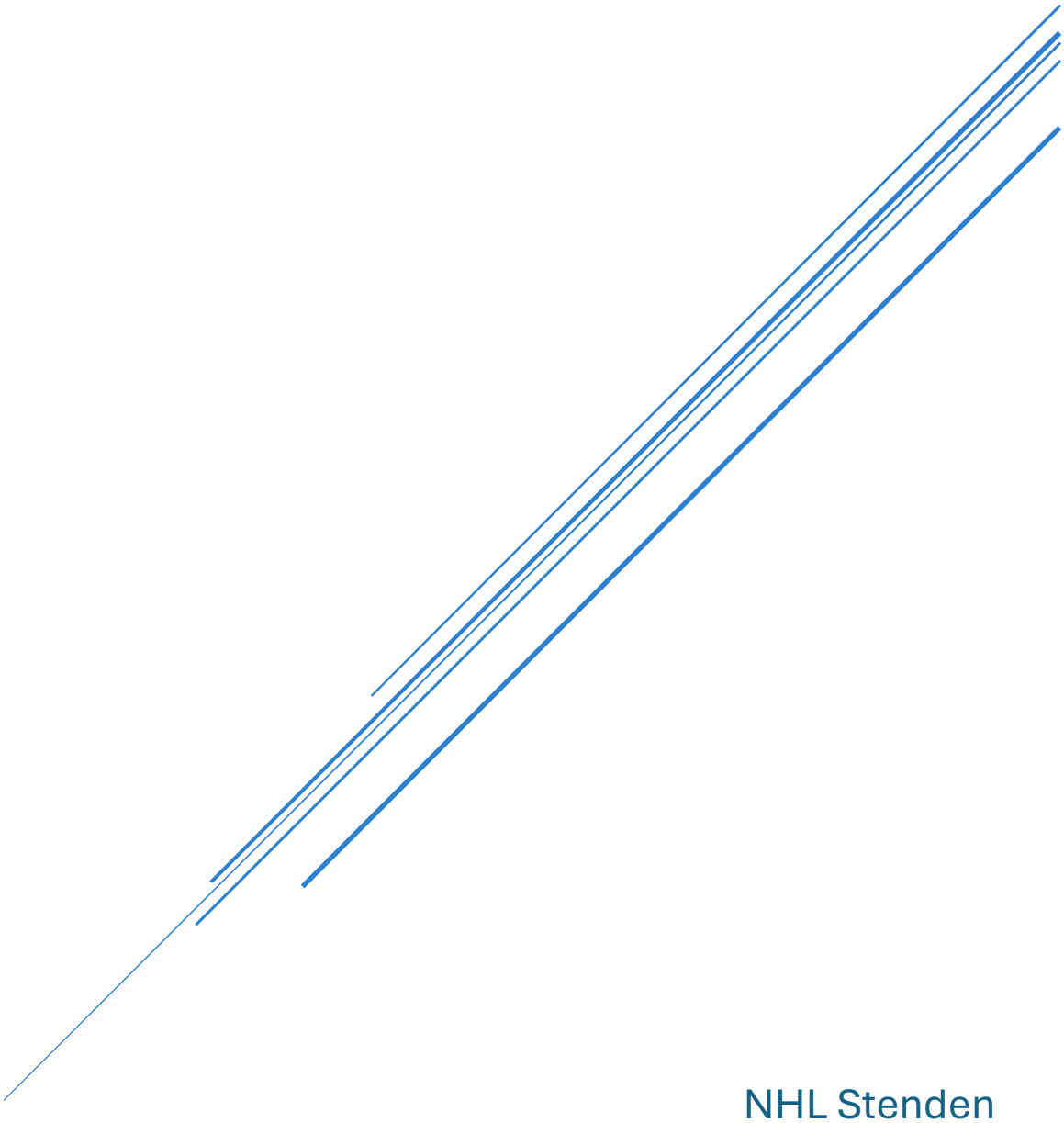
ADVIES DOCUMENT

Jabberpoint



NHL Stenden
Software Quality – Elmedin Arifi & Yunus Karakoc

Contents

Algemeen advies 3
Template Pattern 5
Factory Pattern 6
Command Pattern 7
Herontwerp Diagrammen 8
Class Diagram 8
Template Pattern Class Diagram 10
Factory Pattern Class Diagram 11
Command Pattern Class Diagram 12

Algemeen advies

Zeer verwarrende comments

- Herschrijf de comments zodat ze duidelijk uitleggen wat de code doet.
- Verwijder overbodige of misleidende comments.

Verkeerde indentatie

- Gebruik een code formatter in je IDE:
  - IntelliJ IDEA: Ctrl + Alt + L
  - Eclipse: Ctrl + Shift + F
- Handmatig verbeteren:
  - Houd consistent 4 spaties per indentatie aan (geen tabs).
  - Zorg dat {} correct geplaatst zijn.

Geen this.attribute gebruikt

- Zet this. voor attributen om duidelijk te maken dat het om een instantievariabele gaat.

XMLaccessor.java heeft geen constructor

- Voeg een constructor toe om de klasse correct te kunnen instantiëren.

Hardcode in Style.java

- Vermijd hardcoded waarden, maar blijf binnen de klasse.
- Zet constanten als static final en geef ze duidelijke namen.

Fouten in exception-strings

- Controleer spelling en zorg dat foutmeldingen nuttige informatie bevatten.

Fout in path naar image

- Gebruik een correcte absolute of relatieve pad.

Geen gebruik van packages

- Organiseer bestanden onder com.nhlstenden of een andere logische structuur.

Unittests

- Unittests helpen bij het vroegtijdig opsporen van fouten en het waarborgen van de functionaliteit van de code.
- Ze maken refactoring veiliger, omdat ontwikkelaars snel kunnen controleren of wijzigingen geen onverwachte problemen veroorzaken.
- Unittests verbeteren de onderhoudbaarheid en betrouwbaarheid van het project.

Template Pattern

Het **Template Pattern** wordt gebruikt om de skeletstructuur van een algoritme vast te leggen in een methode, terwijl specifieke stappen van het algoritme door subklassen worden geïmplementeerd. Dit zorgt ervoor dat de algemene volgorde van de stappen wordt gevolgd, maar biedt flexibiliteit voor subklassen om specifieke gedragingen te definiëren.

Waar in Jabberpoint?

1. SlideItem
   a. De draw()-methode is uit Slide gehaald en de render()-methode is geïmplementeerd in SlideItem.
   b. De render()-methode is de Template Method die de algemene structuur van het tekenproces afhandelt.
   c. Elke specifieke inheritor van SlideItem (zoals TextItem of BitmapItem) implementeert zijn eigen versie van de draw()-methode om de specifieke tekenlogica toe te voegen.
   d. De inheritors hoeven alleen de draw()-methode te implementeren, terwijl de algemene render()-methode de volgorde van de stappen vastlegt. Dit zorgt ervoor dat de algemene structuur consistent blijft, terwijl de specifieke tekenstappen flexibel kunnen worden gedefinieerd.

Hoe te implementeren?

- Implementeer een render()-methode in SlideItem die de Template Method is, met de algemene volgorde van de tekenstappen.
- Maak de draw()-methode abstract of gedeeltelijk geïmplementeerd in SlideItem, zodat inheritors hun eigen tekenlogica kunnen invullen.
- De inheritors, zoals TextItem of BitmapItem, vullen de draw()-methode in met hun specifieke tekenlogica, terwijl ze de gemeenschappelijke structuur van render() behouden.

Voordelen van Template Pattern

✓ **Code hergebruik**: De gemeenschappelijke stappen van het tekenproces hoeven niet herhaald te worden in elke inheritor.
✓ **Schaalbaarheid**: Nieuwe SlideItems kunnen eenvoudig worden toegevoegd, en elke nieuwe inheritor volgt dezelfde renderflow zonder dat de code voor de algemene stappen opnieuw geschreven hoeft te worden.
✓ **Verbeterde onderhoudsbaarheid**: Wijzigingen in de algemene structuur van het tekenproces kunnen op één plek (de render**()**-methode) worden doorgevoerd, waardoor duplicatie van code in de inheritors wordt voorkomen.
✓ **Flexibiliteit**: Elke inheritor kan zijn eigen specifieke tekenlogica definiëren, terwijl de algemene flow wordt bewaard.

Factory Pattern

Het Factory Pattern wordt gebruikt om objecten aan te maken zonder dat de aanroepende code expliciet hoeft te weten welke specifieke klasse wordt geïnstantieerd. Dit verhoogt de flexibiliteit en uitbreidbaarheid van de code.

Waar in Jabberpoint?

1. ItemFactory (voor Slide Items)
   a. Momenteel worden TextItem en BitmapItem rechtstreeks aangemaakt.
   b. Een ItemFactory zou dit kunnen centraliseren, waardoor toekomstige uitbreidingen eenvoudiger worden.
   c. De ItemFactory maakt een nieuw SlideItem (zoals TextItem, BitmapItem, of een ander nieuw type) op basis van de vereiste parameters.
   d. Dit centraliseert de creatie van alle SlideItems, waardoor de code overzichtelijker en gemakkelijker uit te breiden is.

Hoe te implementeren?

- Maak een Factory interface met een methode zoals createItem() die een object van een specifieke klasse aanmaakt.
- Maak afzonderlijke fabrieken voor elk type object (bijvoorbeeld StyleFactory, AccessorFactory, ItemFactory).
- In plaats van rechtstreeks een klasse aan te maken, kan de aanroepende code nu een object van de juiste fabriek gebruiken om het juiste type object te verkrijgen.

Voordelen van Factory Pattern

✓ **Minder afhankelijkheden**: De aanroepende code hoeft niet te weten welke specifieke klasse wordt aangemaakt.
✓ **Uitbreidbaarheid**: Nieuwe objecttypen, zoals **VideoItem** of andere **SlideItems**, kunnen eenvoudig worden toegevoegd zonder de bestaande code te wijzigen.
✓ **Flexibiliteit**: Objecten kunnen dynamisch worden gecreëerd op basis van bepaalde parameters, zoals een string of nummer.

Command Pattern

Het Command Pattern wordt gebruikt om acties los te koppelen van hun daadwerkelijke implementatie. Dit is vooral handig wanneer je verschillende invoermethoden (bijvoorbeeld sneltoetsen en menu-opties) wilt laten verwijzen naar dezelfde actie, zonder dat de invoermethoden zelf weten hoe de actie precies wordt uitgevoerd.

Waar in JabberPoint?

1. MenuController en KeyController
   a. Beide bevatten logica om de presentatie te besturen via menu-opties en toetsenbordinvoer.
   b. Acties zoals "volgende slide", "vorige slide" en "afsluiten" worden op meerdere plaatsen in de code herhaald, wat leidt tot duplicatie.
   c. Met het Command Pattern kunnen deze acties in afzonderlijke **command**-klassen worden encapsuleerd, zodat ze hergebruikt kunnen worden in zowel het menu als via toetsenbordbediening, zonder dat de logica opnieuw geschreven hoeft te worden.

Hoe te implementeren?

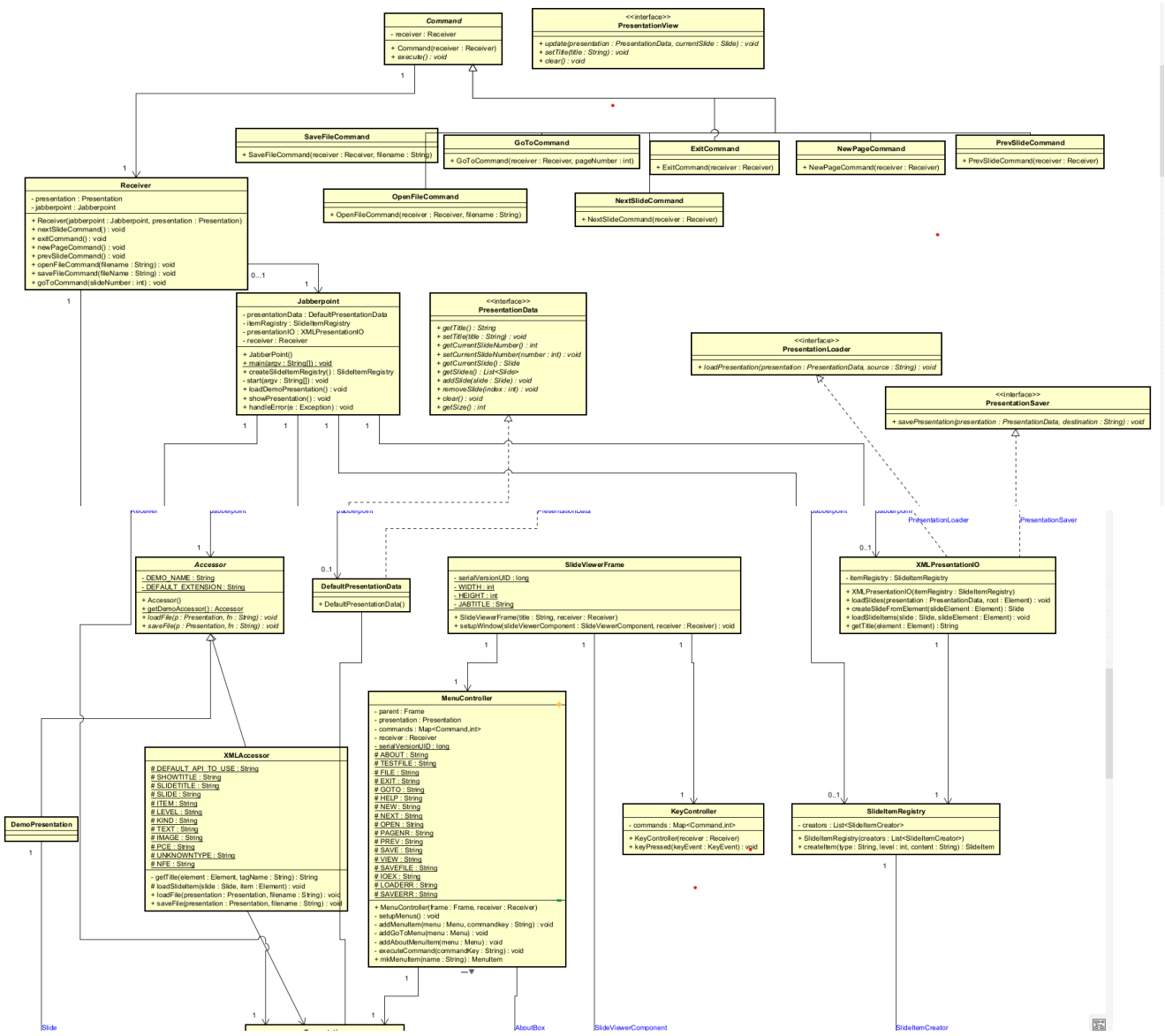
- Maak een Command interface met een execute()-methode die de actie uitvoert.
- Maak voor elke specifieke actie een aparte Command-klasse die de execute()-methode implementeert.
- In KeyController en MenuController worden de toetsen en menu-items gekoppeld aan deze Command-objecten, in plaats van direct methoden aan te roepen. Hierdoor wordt de aanroepende code losgekoppeld van de implementatie van de actie.
- Maak een Receiver-klasse die de echte logica van de actie bevat. De Receiver voert daadwerkelijk de opdracht uit waar het Command-object naar verwijst.

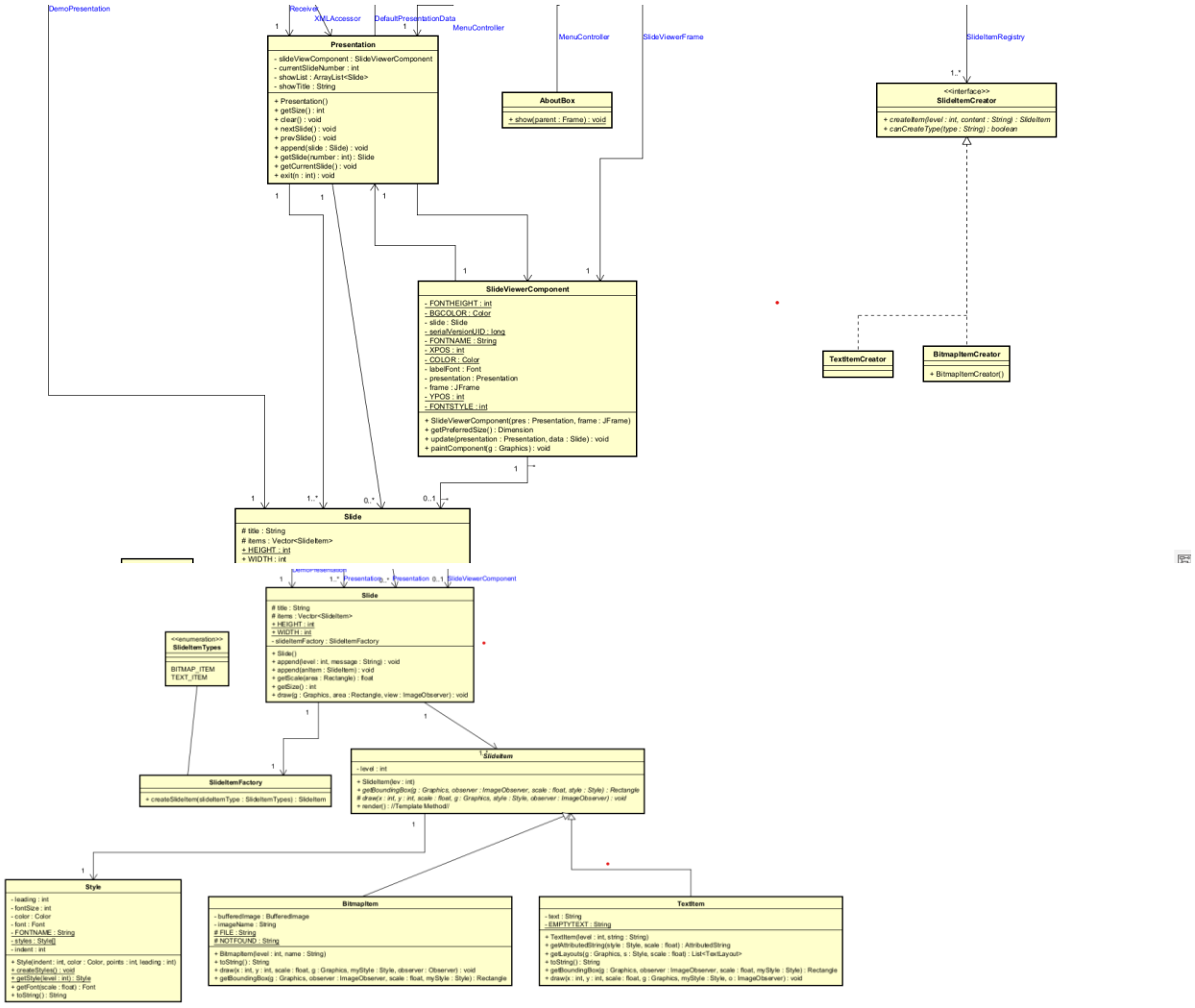
Voordelen van Command Pattern

✓ **Minder duplicatie**: Dezelfde actie kan nu zowel via menu-opties als sneltoetsen worden uitgevoerd, zonder herhaalde code.
✓ **Uitbreidbaarheid**: Nieuwe commando's kunnen eenvoudig worden toegevoegd zonder bestaande code aan te passen, omdat de nieuwe acties simpelweg nieuwe **Command**-klassen kunnen zijn.
✓ **Losse koppeling**: De controllers hoeven niet te weten hoe de acties precies worden uitgevoerd, wat de code flexibeler en gemakkelijker onderhoudbaar maakt.

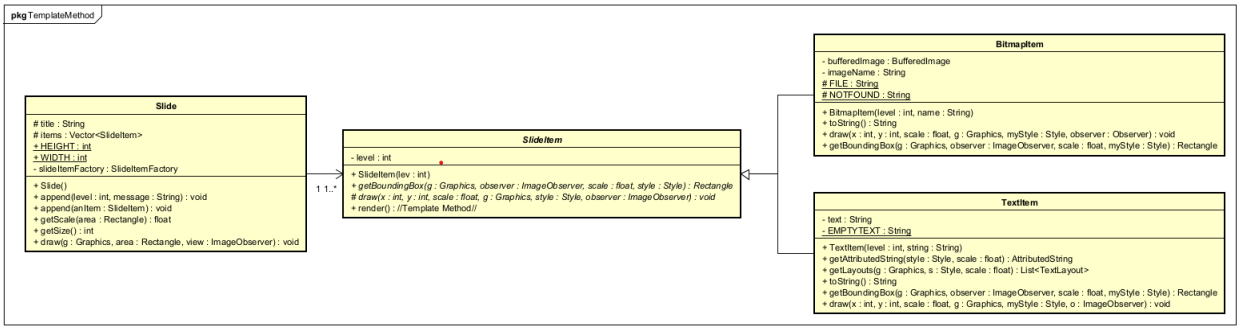
Herontwerp Diagrammen

Class Diagram

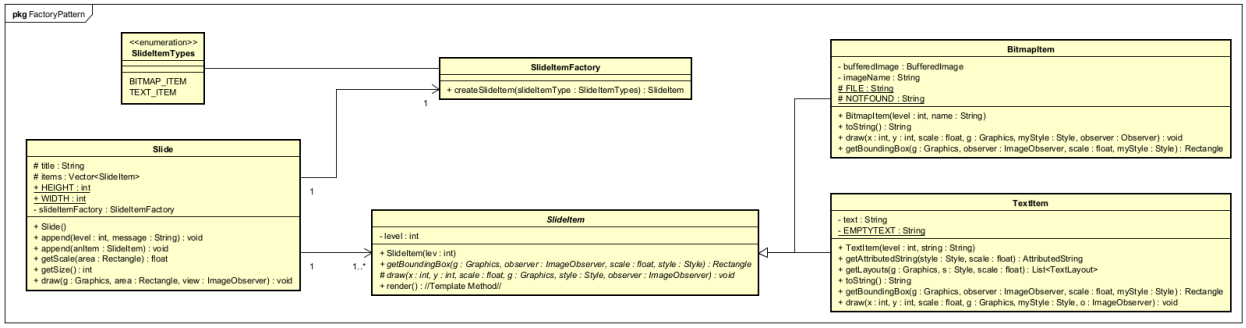



DemoPresentation
Receiver
XMLAccessor
DefaultPresentationData
MenuController
MenuController
SlideViewerFrame
SlideItemRegistry
Presentation
- slideViewComponent : SlideViewerComponent
- currentSlideNumber : int
- showList : ArrayList<Slide>
- showTitle : String
+ Presentation()
+ getSize() : int
+ clear() : void
+ nextSlide() : void
+ prevSlide() : void
+ append(slide : Slide) : void
+ getSlide(number : int) : Slide
+ getCurrentSlide() : void
+ exit(n : int) : void
AboutBox
+ show(parent : Frame) : void
<<interface>>
SlideItemCreator
+ createItem(level : int, content : String) : SlideItem
+ canCreateType(type : String) : boolean
SlideViewerComponent
- FONTHEIGHT : int
- BGCOLOR : Color
- slide : Slide
- serialVersionUID : long
- FONTNAME : String
- XPOS : int
- COLOR : Color
- labelFont : Font
- presentation : Presentation
- frame : JFrame
- YPOS : int
- FONTSTYLE : int
+ SlideViewerComponent(pres : Presentation, frame : JFrame)
+ getPreferredSize() : Dimension
+ update(presentation : Presentation, data : Slide) : void
+ paintComponent(g : Graphics) : void
TextItemCreator
BitmapItemCreator
+ BitmapItemCreator()
Slide
title : String
items : Vector<SlideItem>
+ HEIGHT : int
+ WIDTH : int
1..*
Presentation
0..*
Presentation
0..1
SlideViewerComponent
Slide
title : String
items : Vector<SlideItem>
+ HEIGHT : int
+ WIDTH : int
- slideItemFactory : SlideItemFactory
+ Slide()
+ append(level : int, message : String) : void
+ append(anItem : SlideItem) : void
+ getScale(area : Rectangle) : float
+ getSize() : int
+ draw(g : Graphics, area : Rectangle, view : ImageObserver) : void
<<enumeration>>
SlideItemTypes
BITMAP_ITEM
TEXT_ITEM
SlideItemFactory
+ createSlideItem(slideItemType : SlideItemTypes) : SlideItem
SlideItem
- level : int
+ SlideItem(lev : int)
+ getBoundingBox(g : Graphics, observer : ImageObserver, scale : float, style : Style) : Rectangle
# draw(x : int, y : int, scale : float, g : Graphics, style : Style, observer : ImageObserver) : void
+ render() : //Template Method//
Style
- leading : int
- fontSize : int
- color : Color
- font : Font
- FONTNAME : String
- styles : Style[]
- indent : int
+ Style(indent: int, color : Color, points : int, leading : int)
+ createStyles() : void
+ getStyle(level : int) : Style
+ getFont(scale : float) : Font
+ toString() : String
BitmapItem
- bufferedImage : BufferedImage
- imageName : String
FILE : String
NOTFOUND : String
+ BitmapItem(level : int, name : String)
+ toString() : String
+ draw(x : int, y : int, scale : float, g : Graphics, myStyle : Style, observer : Observer) : void
+ getBoundingBox(g : Graphics, observer : ImageObserver, scale : float, myStyle : Style) : Rectangle
TextItem
- text : String
- EMPTYTEXT : String
+ TextItem(level : int, string : String)
+ getAttributedString(style : Style, scale : float) : AttributedString
+ getLayouts(g : Graphics, s : Style, scale : float) : List<TextLayout>
+ toString() : String
+ getBoundingBox(g : Graphics, observer : ImageObserver, scale : float, myStyle : Style) : Rectangle
+ draw(x : int, y : int, scale : float, g : Graphics, myStyle : Style, o : ImageObserver) : void

Template Pattern Class Diagram



Factory Pattern Class Diagram



Command Pattern Class Diagram