Object Georiënteerd Ontwerpen

Chris Klunder – Object Georiënteerd Ontwerpen – 5051053

Voor het vak Object Georiënteerd Ontwerpen heb ik de opdracht gekregen om de software Jabberpoint te refactoren. Jabberpoint is een presentatie tool die niet echt gezien kan worden als een echt bestaande software. Over het algemeen wordt het gebruikt als educatief hulpmiddel om programmeurs te helpen bij het begrijpen van basisprincipes van het maken van GUI's (Grafische user interface), het laden en weergeven van inhoud, het omgaan met user interactie zoals klikken en het indrukken van toetsen en het opslaan en laden van presentatiebestanden. Dit document zal al mijn gemaakte wijzigingen bevatten met hierbij een korte uitleg waarom deze wijziging is gemaakt. Op de volgende pagina staat een inhoudsopgave. Via deze inhoudsopgave kan gemakkelijk genavigeerd worden naar de verschillende changes die zijn gemaakt.

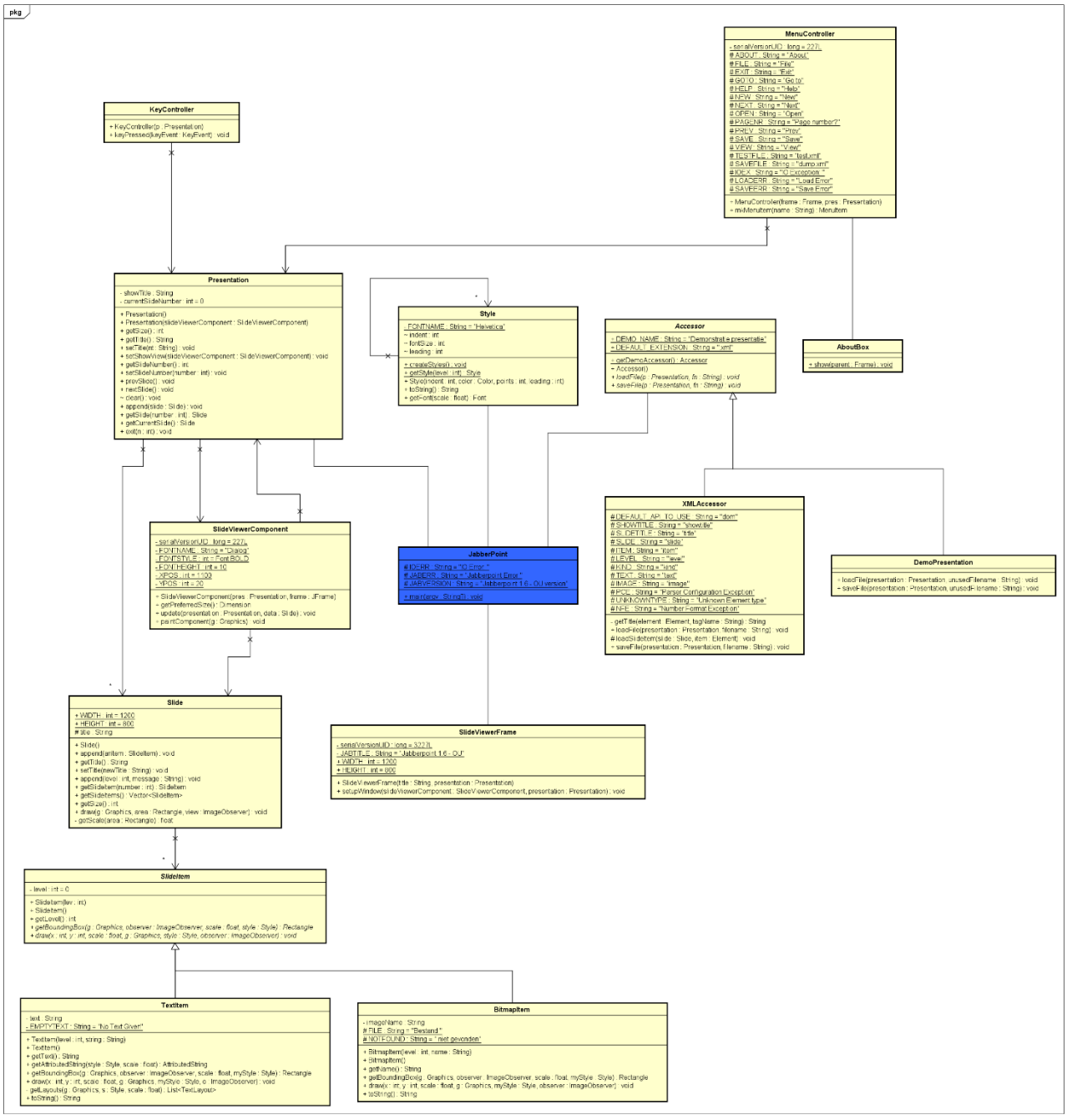
Sommige veranderingen die zijn gemaakt, zijn gemaakt in de gehele applicatie en niet gekoppeld aan een specifieke klasse. Als dit het geval is, wordt dit in het desbetreffende hoofdstuk aangegeven.

Inhoudsopgave

Figuren- en tabellenlijst

1 – Originele klasse diagram

Bij het begin van deze opdracht, is er een klasse diagram gemaakt van de huidige situatie. Hieronder wordt dit klasse diagram weergegeven. De afbeelding is tevens toegevoegd aan de GitHub repository in de map “img”.



Figuur 1 - Originele klasse diagram

2 – Verwijderen van niet gebruikte code

Aan het begin van de opdracht heb ik Jabberpoint doorlopen om te kijken hoe het programma precies in elkaar zit. Hierbij kwam ik er al vrij snel achter dat er veel overbodige / niet gebruikte code in staat. Enkele voorbeelden hiervan zijn niet gebruikte velden in Accessor, TextItem en XMLAccessor en niet gebruikte constructors in BitmapItem en TextItem. Om de code op te schonen, heb ik besloten dergelijke code te verwijderen.

Hieronder vallen echter ook overbodige casts. Denk hierbij aan een volgend scenario:

```
double value = (double) 57.0 / 25.5;
```

In bovenstaand voorbeeld is de variabele al een double, maar er wordt alsnog gecast naar een double door middel van (double).

Een ander voorbeeld van verwijderde code is het setten van waarden van variabelen naar null wanneer deze automatisch al op null geset worden. Hieronder een voorbeeld van de originele code van de SlideViewerComponent klasse.

```
private JFrame frame = null;
```

```
showList = new ArrayList<Slide>();
```

In bovenstaande code is <Slide> overbodig, deze regel in de Presentation klasse is dus veranderd naar

```
showList = new ArrayList<>();
```

3 – Final fields

Ik heb diverse velden in verschillende klassen veranderd naar final velden. Een voorbeeld van zo'n klasse is de Style klasse (er zijn natuurlijk nog meer klassen waarop dit is toegepast). Door het gebruik van final velden, kunnen de waarden hiervan niet meer aangepast worden na de initialisatie van dit veld. Door deze verandering kunnen velden waarbij dit van toepassing is dus niet meer veranderd worden na de initialisatie, waardoor de kwaliteit van de code verbeterd. Hieronder is een kort code fragment weergegeven van hoe dit is toegepast in de Style klasse.

```
private final int indent;
private final Color color;
private final Font font;
private final int fontSize;
private final int leading;
```

4 – Getters in plaats van directe toegang tot velden in Style klasse

```
public Rectangle getBoundingBox(Graphics g, ImageObserver observer, float
scale, Style myStyle) {
    return new Rectangle((int) (myStyle.indent * scale), 0,
          (int) (bufferedImage.getWidth(observer) * scale),
          ((int) (myStyle.leading * scale)) +
          (int) (bufferedImage.getHeight(observer) * scale));
}
```

Bovenstaand wordt de originele code weergegeven van een methode uit de BitmapItem klasse. Hierin wordt de velen indent en leading direct aangeroepen. Deze velden in de Style klasse hebben geen access modifier (public of private), waardoor dit mogelijk is. Een nettere manier om deze velden aan te roepen, is door deze velden op private te zetten en om getters hiervoor te maken. Dit heb ik gedaan, waardoor de methode er nu als volgt uit komt te zien.

```
public Rectangle getBoundingBox(Graphics g, ImageObserver observer, float
scale, Style myStyle) {
    return new Rectangle((int) (myStyle.getIndent() * scale), 0,
          (int) (this.bufferedImage.getWidth(observer) * scale),
          ((int) (myStyle.getLeading() * scale)) +
          (int) (this.bufferedImage.getHeight(observer) * scale));
}
```

Na deze change ben ik direct alle andere klassen bij langsgegaan om alle velden op private te zetten die hier nog niet op waren gezet.

5 – XMLAccessor saveFile methode

```
public void saveFile(Presentation presentation, String filename) throws
IOException {
    PrintWriter out = new PrintWriter(new FileWriter(filename));
    out.println("<?xml version=\"1.0\"?>");
    out.println("<!DOCTYPE presentation SYSTEM \"jabberpoint.dtd\">");
    out.println("<presentation>");
    out.print("<showtitle>");
    out.print(presentation.getTitle());
    out.println("</showtitle>");
    for (int slideNumber=0; slideNumber<presentation.getSize();
slideNumber++) {
       Slide slide = presentation.getSlide(slideNumber);
       out.println("<slide>");
       out.println("<title>" + slide.getTitle() + "</title>");
       Vector<SlideItem> slideItems = slide.getSlideItems();
       for (int itemNumber = 0; itemNumber<slideItems.size(); itemNumber++)
{
          SlideItem slideItem = (SlideItem)
slideItems.elementAt(itemNumber);
          out.print("<item kind=");
          if (slideItem instanceof TextItem) {
             out.print("\"text\" level=\"" + slideItem.getLevel() + "\">");
             out.print( ( (TextItem) slideItem).getText());
          }
          else {
             if (slideItem instanceof BitmapItem) {
                out.print("\"image\" level=\"" + slideItem.getLevel() +
"\">");
                out.print( ( (BitmapItem) slideItem).getName());
             }
             else {
                System.out.println("Ignoring " + slideItem);
             }
          }
          out.println("</item>");
       }
       out.println("</slide>");
    }
    out.println("</presentation>");
    out.close();
}
```

Bovenstaand wordt de originele code weergegeven van de saveFile methode in de XMLAccessor. Zoals kan worden gezien, is deze methode zeer lang en er zijn zeker een aantal verbeterpunten aanwezig. Hieronder staat de gerefactorde code.

```
public void saveFile(Presentation presentation, String filename) throws
IOException {
    PrintWriter out = new PrintWriter(new FileWriter(filename));
    out.println("<?xml version=\"1.0\"?>");
    out.println("<!DOCTYPE presentation SYSTEM \"jabberpoint.dtd\">");
    out.println("<presentation>");
    out.print("<showtitle>" + presentation.getTitle() + "</showtitle>");
    for (Slide slide : presentation.getSlides()) {
       out.println("<slide>");
       out.println("<title>" + slide.getTitle() + "</title>");

       for (SlideItem slideItem : slide.getSlideItems()) {
          if (slideItem instanceof TextItem) {
             out.println("<item kind=\"text\" level=\"" +
slideItem.getLevel() + "\">");
             out.println(((TextItem) slideItem).getText());
             out.println("</item>");
          } else if (slideItem instanceof BitmapItem) {
             out.println("<item kind=\"image\" level=\"" +
slideItem.getLevel() + "\">");
             out.println(((BitmapItem) slideItem).getName());
             out.println("</item>");
          } else {
             System.out.println("Ignoring " + slideItem);
          }
       }

       out.println("</slide>");
    }

    out.println("</presentation>");
    out.close();
}
```

Direct kan worden gezien dat de code al een stuk korter is. Ik heb er namelijk voor gekozen om for each loops te gebruiken in plaats van de voorheen gebruikte for loops, waardoor enkele regels overbodig zijn geworden en zijn weggehaald. Hiervoor zijn ook enkele getters toegevoegd aan de klasse die gebruikt worden in de methode. Daarnaast is er gekozen om een else if te gebruiken in plaat van een else met een if else hierin aan het einde van de methode om de kwaliteit van de code verder te verbeteren.

6 – Toevoegen van this. aan velden

Bij het aanroepen van velden van een klasse binnen deze klasse, werd in de originele versie vrijwel geen this. gebruikt. Het is echter wel netter om dit te doen.

```
public KeyController(Presentation p) {
    presentation = p;
}
```

Door deze verandering veranderde code zoals hierboven in het volgende:

```
public KeyController(Presentation p) {
    this.presentation = p;
}
```

Deze verandering is toegepast op alle klasses waarin dit het geval was.

7 – Fix voor Go To- functionaliteit

Voorheen konden gebruikers door middel van de go to-functionaliteit naar een slide navigeren die groter of kleiner is dan het aantal aanwezige slides. Er kon bijvoorbeeld naar slide 7 van de 5 worden genavigeerd.

Om deze bug op te lossen, heb ik een extra if check toegevoegd aan de methode die hiervoor verantwoordelijk is om dit te checken. De code is nu als volgt:

```
public void setSlideNumber(int number) {
    if (number <= (this.showList.size() - 1) && number >= 0) {
       this.currentSlideNumber = number;
       if (this.slideViewComponent != null) {
          this.slideViewComponent.update(this,
getSlide(this.currentSlideNumber));
       }
    }
}
```

8 – Fix voor Next slide shortcut

In de originele versie kon men navigeren naar de volgende slide door middel van het indrukken van ctrl + n. Dezelfde shortcut werd echter gebruikt voor het aanmaken van een nieuwe presentatie, waardoor problemen ontstonden. Om dit op te lossen, heb ik de shortcut voor het navigeren naar de volgende slide veranderd naar ctrl + shift +n, zodat deze twee functionaliteiten niet meer dezelfde shortcut gebruiken en beide functionaliteiten daadwerkelijk gebruikt kunnen worden.

```
public MenuItem mkMenuItem(String name) {
    return new MenuItem(name, name.equals("Next") ? new
MenuShortcut(name.charAt(0), true) : new MenuShortcut(name.charAt(0)));
}
```

In de bovenstaande methode wordt nu gecheckt of het menu item “Next” is. Als dit het geval is, wordt bij het aanmaken van de MenuShortcut voor useShiftModifier de value true meegegeven, zodat deze shortcut gebruikt kan worden door middel van de toetsen ctrl + shift + n. De shortcut van de andere functionaliteiten blijft ctrl + <<toets>>.

9 -Bloater MenuController

```
public MenuController(Frame frame, Presentation pres) {
    parent = frame;
    presentation = pres;
    MenuItem menuItem;
    Menu fileMenu = new Menu(FILE);
    fileMenu.add(menuItem = mkMenuItem(OPEN));
    menuItem.addActionListener(new ActionListener() {
       public void actionPerformed(ActionEvent actionEvent) {
          presentation.clear();
          Accessor xmlAccessor = new XMLAccessor();
          try {
             xmlAccessor.loadFile(presentation, TESTFILE);
             presentation.setSlideNumber(0);
          } catch (IOException exc) {
             JOptionPane.showMessageDialog(parent, IOEX + exc,
                  LOADERR, JOptionPane.ERROR_MESSAGE);
          }
          parent.repaint();
       }
    } );
    fileMenu.add(menuItem = mkMenuItem(NEW));
    menuItem.addActionListener(new ActionListener() {
       public void actionPerformed(ActionEvent actionEvent) {
          presentation.clear();
          parent.repaint();
       }
    });
    fileMenu.add(menuItem = mkMenuItem(SAVE));
    menuItem.addActionListener(new ActionListener() {
       public void actionPerformed(ActionEvent e) {
          Accessor xmlAccessor = new XMLAccessor();
          try {
             xmlAccessor.saveFile(presentation, SAVEFILE);
          } catch (IOException exc) {
             JOptionPane.showMessageDialog(parent, IOEX + exc,
                   SAVEERR, JOptionPane.ERROR_MESSAGE);
          }
       }
    });
    fileMenu.addSeparator();
    fileMenu.add(menuItem = mkMenuItem(EXIT));
    menuItem.addActionListener(new ActionListener() {
       public void actionPerformed(ActionEvent actionEvent) {
          presentation.exit(0);
       }
    });
    add(fileMenu);
    Menu viewMenu = new Menu(VIEW);
    viewMenu.add(menuItem = mkMenuItem(NEXT));
    menuItem.addActionListener(new ActionListener() {
       public void actionPerformed(ActionEvent actionEvent) {
          presentation.nextSlide();
       }
    });
    viewMenu.add(menuItem = mkMenuItem(PREV));
    menuItem.addActionListener(new ActionListener() {
       public void actionPerformed(ActionEvent actionEvent) {
          presentation.prevSlide();
```

```
        }
    });
    viewMenu.add(menuItem = mkMenuItem(GOTO));
    menuItem.addActionListener(new ActionListener() {
       public void actionPerformed(ActionEvent actionEvent) {
          String pageNumberStr =
JOptionPane.showInputDialog((Object)PAGENR);
          int pageNumber = Integer.parseInt(pageNumberStr);
          presentation.setSlideNumber(pageNumber - 1);
       }
    });
    add(viewMenu);
    Menu helpMenu = new Menu(HELP);
    helpMenu.add(menuItem = mkMenuItem(ABOUT));
    menuItem.addActionListener(new ActionListener() {
       public void actionPerformed(ActionEvent actionEvent) {
          AboutBox.show(parent);
       }
    });
    setHelpMenu(helpMenu);     // nodig for portability (Motif, etc.).
}
```

De bovenstaande code geeft de originele code weer van de MenuController constructor. Deze constructor is 77 regels aan code, dus is hier sprake van een Bloater. De meest voorkomende oplossing voor een bloater is het afsplitsen van code in verschillende methoden. Dit is dus precies wat ik heb gedaan.

```
public MenuController(Frame frame, Presentation pres) {
    this.parent = frame;
    this.presentation = pres;
    buildFileMenu();
    buildViewMenu();
    buildHelpMenu();
}

/**
 * Method to create a menu item including its functionalities
 * @param menuItem The menu item
 * @param method The functionalities the menu item should have
 * @return The menu item
 */
private MenuItem createMenuItem(MenuItem menuItem, Runnable method) {
    menuItem.addActionListener(e -> method.run());
    return menuItem;
}

/**
 * Method to create a shortcut for menu items
 * @param name The name of the menu item
 * @return The menu item
 */
public MenuItem mkMenuItem(String name) {
    return new MenuItem(name, name.equals("Next") ? new
MenuShortcut(name.charAt(0), true) : new MenuShortcut(name.charAt(0)));
}
```

```
/**
 * Method to save a presentation file
 */
private void saveFile() {
    Accessor xmlAccessor = new XMLAccessor();
    try {
       xmlAccessor.saveFile(presentation, MenuLabels.SAVEFILE);
    } catch (IOException exc) {
       JOptionPane.showMessageDialog(parent, MenuLabels.IOEX + exc,
             MenuLabels.SAVEERR, JOptionPane.ERROR_MESSAGE);
    }
}

/**
 * Method to ope a presentation, in this case the test presentation
 */
private void openPresentation() {
    presentation.clear();
    Accessor xmlAccessor = new XMLAccessor();
    try {
       xmlAccessor.loadFile(presentation, MenuLabels.TESTFILE);
       presentation.setSlideNumber(0);
       parent.repaint();
    } catch (IOException ex) {
       JOptionPane.showMessageDialog(parent, MenuLabels.IOEX + ex,
MenuLabels.LOADERR, JOptionPane.ERROR_MESSAGE);
    }
}

/**
 * Method to create a new presentation
 */
private void newPresentation() {
    presentation.clear();
    parent.repaint();
}

/**
 * Build the file menu with all its items
 */
private void buildFileMenu() {
    Menu fileMenu = new Menu(MenuLabels.FILE);
    fileMenu.add(createMenuItem(mkMenuItem(MenuLabels.OPEN),
this::openPresentation));
    fileMenu.add(createMenuItem(mkMenuItem(MenuLabels.NEW),
this::newPresentation));
    fileMenu.add(createMenuItem(mkMenuItem(MenuLabels.SAVE),
this::saveFile));
    fileMenu.add(createMenuItem(mkMenuItem(MenuLabels.EXIT), () ->
presentation.exit(0)));
    add(fileMenu);
}

/**
 * Get the slide number the user has entered
 * @return The slide number as integer
 */
private int getSlideNumberFromInput() {
    String slideNumber = JOptionPane.showInputDialog(MenuLabels.PAGENR);
    return Integer.parseInt(slideNumber);
```

```
}

/**
 * Method to build the view menu with all its items
 */
private void buildViewMenu() {
    Menu viewMenu = new Menu(MenuLabels.VIEW);
    viewMenu.add(createMenuItem(mkMenuItem(MenuLabels.NEXT),
presentation::nextSlide));
    viewMenu.add(createMenuItem(mkMenuItem(MenuLabels.PREV),
presentation::prevSlide));
    viewMenu.add(createMenuItem(mkMenuItem(MenuLabels.GOTO), () ->
presentation.setSlideNumber(getSlideNumberFromInput() - 1)));
    add(viewMenu);
}

/**
 * Method to build the help menu with all its items
 */
private void buildHelpMenu() {
    Menu helpMenu = new Menu(MenuLabels.HELP);
    helpMenu.add(createMenuItem(mkMenuItem(MenuLabels.ABOUT), () ->
AboutBox.show(parent)));
    setHelpMenu(helpMenu);
}
```

Dit is de nieuwe code van de menu controller. Hierin heb ik het bouwen van de verschillende menu's opgesplitst in aparte methoden en heb ik ook de methoden die aangeroepen werden in de constructor afgesplitst. Hierdoor is de overzichtelijkheid van de klasse aanzienlijk verbeterd en is de code een stuk eenvoudiger te begrijpen. Daarnaast is de bloater nu verholpen.

10 – Aparte klasse voor MenuLabels & XMLAccessorLabels

De MenuController bevatte in eerste instantie veel fields, zoals hieronder weergegeven:

```
private Frame parent; // het frame, alleen gebruikt als ouder voor de
Dialogs
private Presentation presentation; // Er worden commando's gegeven aan de
presentatie

private static final long serialVersionUID = 227L;

protected static final String ABOUT = "About";
protected static final String FILE = "File";
protected static final String EXIT = "Exit";
protected static final String GOTO = "Go to";
protected static final String HELP = "Help";
protected static final String NEW = "New";
protected static final String NEXT = "Next";
protected static final String OPEN = "Open";
protected static final String PAGENR = "Page number?";
protected static final String PREV = "Prev";
protected static final String SAVE = "Save";
protected static final String VIEW = "View";

protected static final String TESTFILE = "test.xml";
protected static final String SAVEFILE = "dump.xml";

protected static final String IOEX = "IO Exception: ";
protected static final String LOADERR = "Load Error";
protected static final String SAVEERR = "Save Error";
```

Hierbij nemen vooral de protected static final String fields veel ruimte in. Hier is in dit geval dan ook sprake van een Large Class. Om dit probleem te verhelpen, heb ik ervoor gekozen om deze velden af te splitsen naar een aparte klasse genaamd MenuLabels. Vervolgens heb ik zowel de MenuControllers and de MenuLabels samenin een package gezet, genaamd Menu. Hierdoor kan ik door één simpele import gebruik maken van alle labels uit MenuLabels in de MenuController.

Hetzelfde principe heb ik toegepast op XMLAccessor. Hier is nu ook een aparte klasse genamd XMLAccessorLabels voor aangemaakt en beide klassen zijn samen in een package gezet genaamd Accessors.

11 – Toevoegen van de StyleFactory

In het begin werden verschillende styles aangemaakt in de Style klasse zelf. Dit is niet de bedoeling. Er is daarom ervoor gekozen om hiervoor een factory aan te maken, genaamd de StyleFactory. Deze factory handelt het aanmaken van de verschillende styles af en bevat ook de methode die gebruikt wordt in andere klassen om de geschikte style te krijgen uit de lijst met aangemaakte styles. Hieronder wordt de StyleFactory klasse weergegeven.

```
public class StyleFactory
{
    private static Style[] styles;

    public static Style createStyle(int indent, Color color, int points,
int leading)
    {
        return new Style(indent, color, points, leading);
    }
    public static void createStyles() {
        styles = new Style[5];
        styles[0] = createStyle(0, Color.red,   48, 20);
        styles[1] = createStyle(20, Color.blue,  40, 10);
        styles[2] = createStyle(50, Color.black, 36, 10);
        styles[3] = createStyle(70, Color.black, 30, 10);
        styles[4] = createStyle(90, Color.black, 24, 10);
    }

    public static Style getStyle(int level) {
        if (level >= styles.length) {
            level = styles.length - 1;
        }
        return styles[level];
    }
}
```

Om de Factory nog netter te maken, heb ik er ook voor gekozen om een aparte methode aan te maken die het aanmaken van de styles afhandelt in createStyles. Op deze manier hoeft er niet bij elke style gebruik gemaakt worden van new Style zoals hieronder weergegeven.

```
styles[0] = new Style(…);
```

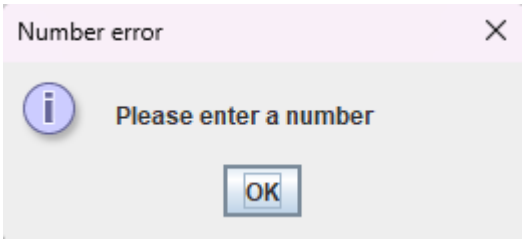
12 – Fix voor geen entry bij go to-functionaliteit

Wanneer een gebruiker geen getal invoerde bij de go to-functionaliteit, kwam de volgende error in de console te staan:

```
Bestand JabberPoint.jpg niet gevonden
Exception in thread "AWT-EventQueue-0" java.lang.NumberFormatException Create breakpoint : For input string: ""
    at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:67)
    at java.base/java.lang.Integer.parseInt(Integer.java:672)
    at java.base/java.lang.Integer.parseInt(Integer.java:778)
    at MenuController$7.actionPerformed(MenuController.java:112)
    at java.desktop/java.awt.MenuItem.processActionEvent(MenuItem.java:692)
    at java.desktop/java.awt.MenuItem.processEvent(MenuItem.java:651)
    at java.desktop/java.awt.MenuComponent.dispatchEventImpl(MenuComponent.java:378)
```

Figuur 2 - Console error go to-functionaliteit

Om deze error op te lossen, heb ik besloten een pop-up venster te tonen wanneer de gebruiker geen getal invoert. Deze pop-up ziet er als volgt uit:



Figuur 3 - Toegevoegde pop-up

De code ziet er nu als volgt uit:

```
private int getSlideNumberFromInput() {
    try {
       String slideNumber = JOptionPane.showInputDialog(PAGENR);
       if (Integer.parseInt(slideNumber) != 0) {
          return Integer.parseInt(slideNumber);
       }
    } catch (Exception ex) {
       JOptionPane.showMessageDialog(parent,
             "Please enter a number!",
             "Number error",
             JOptionPane.INFORMATION_MESSAGE
       );
    }
       return 0;
   }
```

13 – Verplaatsen van exit methode naar de JabberPoint klasse

Oorspronkelijk stond de exit methode om de applicatie af te sluiten in de Presentation klasse. Deze klasse is echter niet de hoofdklasse van de applicatie. Om deze reden heb ik ervoor gekozen om deze methode te verplaatsen naar de JabberPoint klasse, omdat dit wel de hoofdklasse is.

```
public static void exit(int n) {
    System.exit(n);
}
```

Ik heb de methode wel veranderd in een static methode, zodat ik deze eenvoudig met behulp van een import statement kan gebruiken in de verschillende klassen.

De methode wordt nu door een import aangeroepen in de verschillende klassen. Onderstaan voorbeeld is code uit de MenuController klasse:

```
import JabberPoint.*;
```

```
fileMenu.add(createMenuItem(mkMenuItem(EXIT), () -> JabberPoint.exit(0)));
```

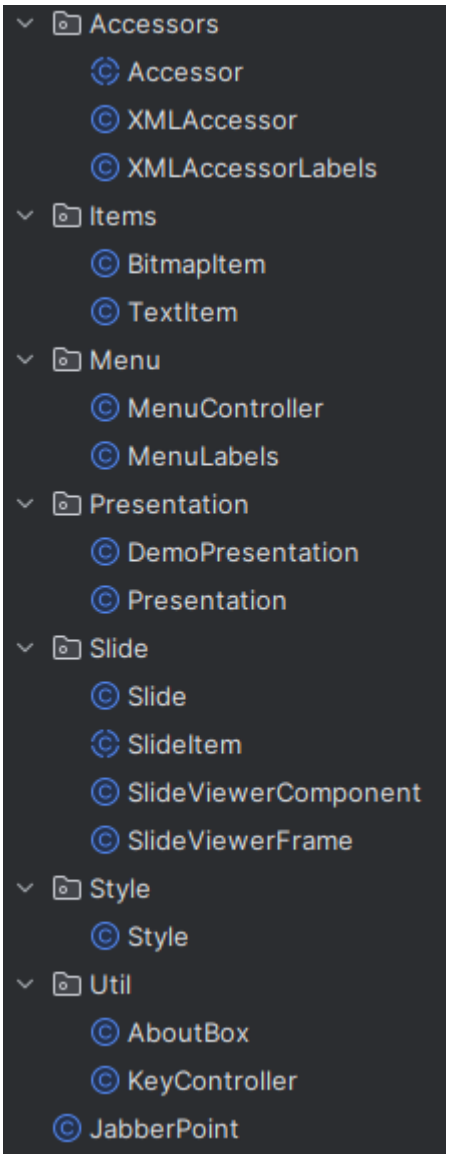
14 – Toevoegen van commentaar

In de aangeleverde versie van JabberPoint stond vrijwel geen commentaar. Af en toe was er een korte regel toegevoegd aan regels code voor het verduidelijken van velden en dergelijke, maar hierbij bleef het. Om de code duidelijker te maken, heb ik besloten om overal waar het door mij nodig werd geacht code commentaar toe te voegen. Door dit commentaar wordt de code een stuk duidelijker voor andere gebruikers, omdat men vrijwel direct kan zien waarvoor welke methoden dienen.

Er is enkel commentaar toegevoegd bij methoden die dit nodig hadden, getters en setters zijn in de meeste gevallen niet voorzien van commentaar.

15 – Toevoegen van Packages

Zoals al in Hoofdstuk 10 – Aparte klasse voor MenuLabels & XMLAccessorLabels genoemd, zijn er packages toegevoegd genaamd Menu en Accessors. Er zijn echter ook packages toegevoegd voor andere delen van de applicatie. In onderstaande afbeelding worden de verschillende packages weergegeven met de klassen die hierin zijn opgenomen.

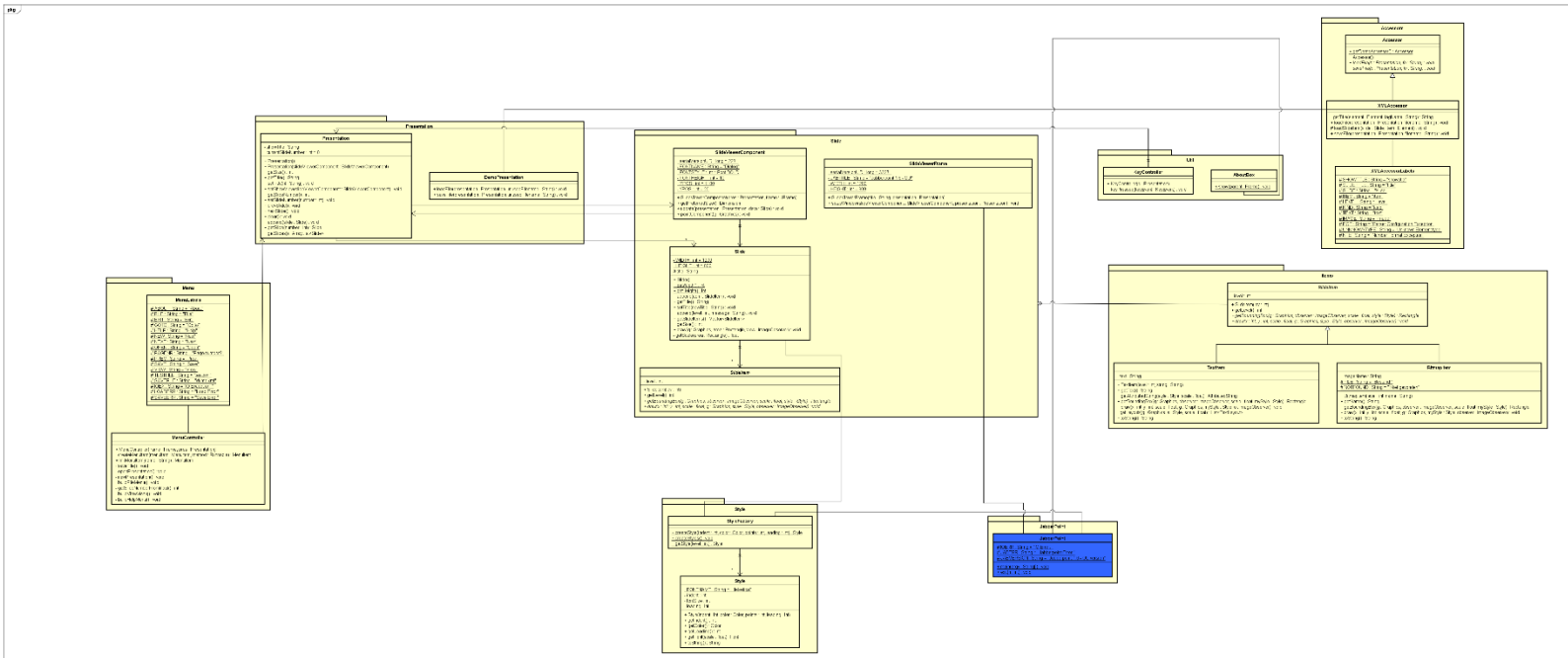


Figuur 4 – Packages

De bovenstaande vond ik persoonlijk het meest logisch, omdat de verschillende delen grotendeels bij elkaar staan in een package, waardoor het eenvoudiger wordt om een klasse terug te vinden.

16 – Nieuwe klasse diagram

Onderstaand is het nieuwe klasse diagram te zien. Dit klasse diagram is gemaakt op basis van alle gemaakte veranderingen in de originele code van JabberPoint. Deze afbeelding is net als de afbeelding van het originele klasse diagram opgenomen in de GitHub repository in de map “img”.



Figuur 5 - Nieuwe klasse diagram