JabberPoint advies.

Kevin Spalink & Aman Trechsel – Software Quality – 26/02/2024

Tekortkomingen

In dit hoofdstuk wordt beschreven welke punten verbeterd kunnen worden binnen het systeem. Dit zijn punten die belangrijk zijn voor de kwaliteit van de software.

Comments

Er zijn veel onderdelen van de code niet voorzien van commentaar. Als er wel commentaar wordt meegegeven is deze vaak niet beschrijvend en heeft het geen toegevoegde waarde. Het schrijven van duidelijke en omschrijvende code is cruciaal, vooral voor het duidelijker maken van complexe delen. Het is essentieel voor de algehele kwaliteit en onderhoudbaarheid van een project.

In dit voorbeeld wordt bijvoorbeeld een complexe functionaliteit afgedaan met 1 niet beschrijvende zin, terwijl er veel meer uitleg kan worden gegeven.

```
// geef de AttributedString voor het item
    public AttributedString getAttributedString(Style style, float scale) {
        AttributedString attrStr = new AttributedString(getText());
        attrStr.addAttribute(TextAttribute.FONT, style.getFont(scale), 0, text.length());
        return attrStr;
    }

// geef de bounding box van het item
    public Rectangle getBoundingBßox(Graphics g, ImageObserver observer,
            float scale, Style myStyle) {
        List<TextLayout> layouts = getLayouts(g, myStyle, scale);
        int xsize = 0, ysize = (int) (myStyle.leading * scale);
        Iterator<TextLayout> iterator = layouts.iterator();
        while (iterator.hasNext()) {
            TextLayout layout = iterator.next();
            Rectangle2D bounds = layout.getBounds();
            if (bounds.getWidth() > xsize) {
                xsize = (int) bounds.getWidth();
            }
            if (bounds.getHeight() > 0) {
                ysize += bounds.getHeight();
            }
            ysize += layout.getLeading() + layout.getDescent();
        }
        return new Rectangle((int) (myStyle.indent*scale), 0, xsize, ysize );
    }
```

Code Kwaliteit

Veel methoden, velden, parameters, variabelen en constanten hebben namen die niet goed beschrijvend zijn of inconsistent. Dit zorgt voor verwarring en maakt het begrijpen van de code onnodig moeilijk. Het is daarom belangrijk om aandacht te besteden aan het kiezen van passende en consistente namen.

Bijvoorbeeld in MenuController wordt er gebruik gemaakt van een ‘mkMenuItem’ methode terwijl in Style een ‘createStyles’ methode gebruikt wordt. Bij de methode van MenuController staat mk voor *make*, wat niet gelijk duidelijk is uit de naam. Ook is het gebruik van *make* en *create*, twee synoniemen, niet consistent gebruik van termen. Het gebruik van werkwoorden in methodes, voornamelijk het eerste woord in een methode, moet consistent zijn.

```
public MenuItem mkMenuItem(String name) {
  return new MenuItem(name, new MenuShortcut(name.charAt(index:0)));
}
```

```
public static void createStyles() {
  styles = new Style[5];
  // De styles zijn vast ingecodeerd.
  styles[0] = new Style(indent:0, Color.red,   points:48, leading:20);  // style voor item-level 0
  styles[1] = new Style(indent:20, Color.blue,  points:40, leading:10); // style voor item-level 1
  styles[2] = new Style(indent:50, Color.black, points:36, leading:10); // style voor item-level 2
  styles[3] = new Style(indent:70, Color.black, points:30, leading:10); // style voor item-level 3
  styles[4] = new Style(indent:90, Color.black, points:24, leading:10); // style voor item-level 4
}
```

Veel methodes ontbreken de '@Override' headers terwijl ze geïmplementeerd zijn vanuit een superclass, wat ervoor zorgt dat de code minder duidelijk is. Het consistent gebruiken van deze headers verbetert de onderhoudbaarheid van de code.

Bijvoorbeeld de methodes ‘getBoundingBox’ en ‘draw’ in BitmapItem zijn geïmplementeerd uit de superclass SlideItem. Alleen deze methodes missen de ‘@Override’ header.

```
// geef de bounding box van de afbeelding
  public Rectangle getBoundingBox(Graphics g, ImageObserver observer, float scale, Style myStyle) {
    return new Rectangle((int) (myStyle.indent * scale), y:0,
        (int) (bufferedImage.getWidth(observer) * scale),
        ((int) (myStyle.leading * scale)) +
        (int) (bufferedImage.getHeight(observer) * scale));
  }

// teken de afbeelding
  public void draw(int x, int y, float scale, Graphics g, Style myStyle, ImageObserver observer) {
    int width = x + (int) (myStyle.indent * scale);
    int height = y + (int) (myStyle.leading * scale);
    g.drawImage(bufferedImage, width, height,(int) (bufferedImage.getWidth(observer)*scale),
                (int) (bufferedImage.getHeight(observer)*scale), observer);
  }
```

Het gebruik van een Vector voor Slide en XMLAccessor is onnodig, aangezien er geen goede reden is om deze type collection te gebruiken. Het is belangrijk om de keuze voor datastructuren te rechtvaardigen op basis van de specifieke vereisten en eigenschappen van de gegevens, om inefficiënties te voorkomen.

Bijvoorbeeld de ‘items’ veld in Slide is een Vector van SlideItem. Dit is onnodig gezien de functie van het veld is het opslaan van een collectie van SlideItems. Hierin zou het gebruik van een HashSet of ArrayList, of zelfs een standaard Array veel logischer en code-efficiënter zijn.

```
public class Slide {
  public final static int WIDTH = 1200;
  public final static int HEIGHT = 800;
  protected String title; // de titel wordt apart bewaard
  protected Vector<SlideItem> items; // de slide-items worden in een Vector bewaard
```

Veel indentaties zijn inconsistent of zijn van slechte kwaliteit, wat kan leiden tot verminderde leesbaarheid en begrijpelijkheid van de code. Het gebruik van consistente en kwalitatieve indentaties is nodig voor goed gestructureerde en onderhoudbare software.

Bijvoorbeeld in de ‘getLayouts’ methode van TextItem is er een willekeurige indentatie op de 3e regel van de methode. Deze indentatie heeft een functie en is alleen maar verwarrend.

```
private List<TextLayout> getLayouts(Graphics g, Style s, float scale) {
  List<TextLayout> layouts = new ArrayList<TextLayout>();
  AttributedString attrStr = getAttributedString(s, scale);
    Graphics2D g2d = (Graphics2D) g;
    FontRenderContext frc = g2d.getFontRenderContext();
    LineBreakMeasurer measurer = new LineBreakMeasurer(attrStr.getIterator(), frc);
    float wrappingWidth = (Slide.WIDTH - s.indent) * scale;
    while (measurer.getPosition() < getText().length()) {
      TextLayout layout = measurer.nextLayout(wrappingWidth);
      layouts.add(layout);
    }
    return layouts;
}
```

Design Patterns

Singleton

- De MenuController wordt slechts eenmaal gebruikt binnen het programma, er is geen reden voor meerdere instanties van de MenuController. Door hier een singleton van te maken zorgt het ervoor dat de MenuController maar één keer gemaakt kan worden.

- Voor KeyController geld precies hetzelfde als voor MenuController.

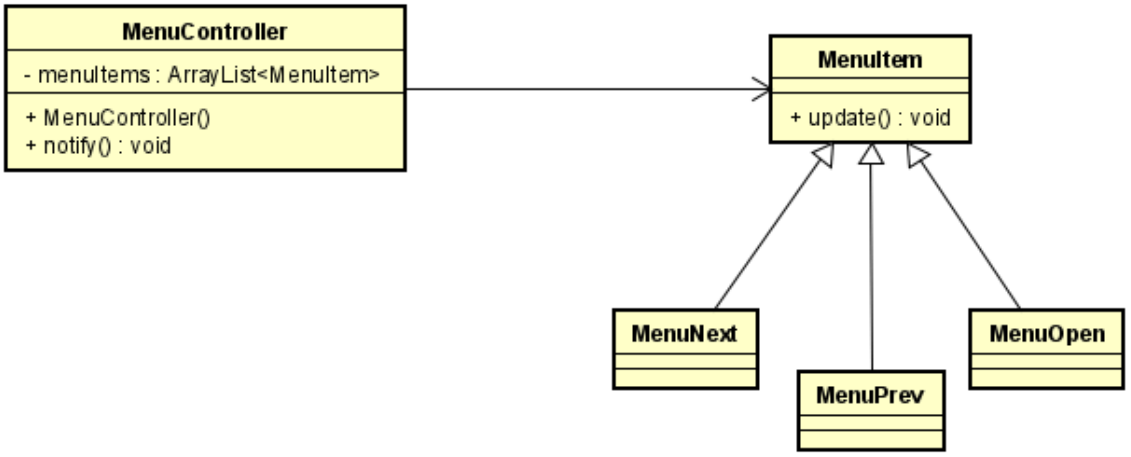
Observer Pattern

- Door het Observer Pattern te gebruiken, kunnen we een betere relatie opzetten tussen de SlideViewerComponent en Presentation. Hierdoor kunnen we de functionaliteit van beide onderdelen scheiden en een nettere en makkelijker uitbreidbare codebasis creëren. Met dit patroon kunnen we ook makkelijker nieuwe manieren toevoegen om de Presentation en de SlideViewerComponent met elkaar te laten werken, waardoor de samenwerking tussen deze twee belangrijke onderdelen van JabberPoint wordt verbeterd.

Command Pattern

- Het Command Pattern is een manier om op een nette manier input te verwerken. Het zet acties in aparte command objects, wat handig is omdat je dan nieuwe manieren om input te geven makkelijk kan toevoegen zonder veel te moeten veranderen aan de bestaande code. Hierdoor kan de KeyController beter groeien en wordt het makkelijker om dingen toe te voegen of aan te passen naarmate het project groter wordt.

- De MenuController heeft momenteel een grote constructor met veel herhaalde code, wat het moeilijk maakt om te begrijpen en te onderhouden. Maar door het Command Pattern te gebruiken en de MenuController om te zetten naar een systeem van met command objects, kunnen we de constructor vereenvoudigen en de herhaalde code verminderen. Hierdoor wordt de code meer modulair en makkelijker te onderhouden, en kan de MenuController efficiënter communiceren met zijn ActionListeners. Met deze verandering wordt niet alleen de code leesbaarder, maar wordt het ook flexibeler om nieuwe functies, of bestaande functies uit de KeyController toe te voegen aan de menu functionaliteit van JabberPoint.



Facade Pattern

- Door deze pattern te gebruiken voor de '*DemoPresentation*', maken we een eenvoudige interface die de complexiteit verbergt. Dit maakt het gemakkelijker om de code uit te breiden, opnieuw te gebruiken en te onderhouden. Ook kan de facade kan opnieuw worden gebruikt voor demos van verschillende delen van de applicatie. Het draagt bij aan een betere modulariteit en gebruiksgemak van de applicatie.

```
class DemoPresentation extends Accessor {

  public void loadFile(Presentation presentation, String unusedFilename) {
    presentation.setTitle(nt:"Demo Presentation");
    Slide slide;
    slide = new Slide();
    slide.setTitle(newTitle:"JabberPoint");
    slide.append(level:1, message:"Het Java Presentatie Tool");
    slide.append(level:2, message:"Copyright (c) 1996-2000: Ian Darwin");
```

Bugs

Slide openen met een niet bestaande afbeelding

Als een slide wordt geopend waar een afbeelding in wordt meegegeven die niet bestaat, dan wordt een witte pagina getoond en wordt er één grote foutmelding gelogd.

```
Bestand JabberPoint.jpg niet gevonden
Exception in thread "AWT-EventQueue-0" java.lang.NullPointerException Create breakpoint : Cannot invoke "java.
    at BitmapItem.draw(BitmapItem.java:64)
    at Slide.draw(Slide.java:73)
    at SlideViewerComponent.paintComponent(SlideViewerComponent.java:71)
    at java.desktop/javax.swing.JComponent.paint(JComponent.java:1128)
    at java.desktop/javax.swing.JComponent.paintChildren(JComponent.java:961)
    at java.desktop/javax.swing.JComponent.paint(JComponent.java:1137)
    at java.desktop/javax.swing.JComponent.paintChildren(JComponent.java:961)
    at java.desktop/javax.swing.JComponent.paint(JComponent.java:1137)
    at java.desktop/javax.swing.JLayeredPane.paint(JLayeredPane.java:586)
    at java.desktop/javax.swing.JComponent.paintChildren(JComponent.java:961)
    at java.desktop/javax.swing.JComponent.paintToOffscreen(JComponent.java:5325)
```

In plaats van alleen een fout te loggen en een witte pagina te tonen kan er ook worden gekozen om niet gevonden mediabestanden te vervangen met een dummy afbeelding die aantoont dat de meegegeven media item niet kan worden gevonden zoals hieronder:

Sommige shortcuts hebben dezelfde toets

De manier waarop shortcuts aangemaakt worden is door de ‘mkMenuItem’ methode in MenuController, waarin de eerste letter van de menu item naam wordt gebruikt. In het geval van “New” en “Next” is dit allebei de letter N (dus CTRL+N). Het is dus belangrijk om eerst te controleren of een shortcut al bestaat, en dan Shift of Alt te gebruiken of combineren voor nieuwe sneltoetsen. Om het schaalbaar te maken zou het beter zijn om deze shortcuts zelf te definiëren in plaats van dynamisch de eerste letter te pakken.

Missende Componenten

Binnen de applicatie zijn meerdere componenten niet aanwezig. Zo wordt er niet gewerkt met packages, maar worden alle bestanden op 1 plek bewaard.

Er zijn geen packages aanwezig

Het gebruik van packages is belangrijk voor de organisatie en structuur van de software. Het maakt het vinden van code die met elkaar te maken hebben gemakkelijker.

Veel fields of methodes hebben geen access modifiers

Fields zoals ‘indent’ en ‘color’ in ‘Style’ hebben geen access modifier. Dit is belangrijk voor veiligheid en concreetheid.

Veel data mist validatie

Methodes zoals het aanmaken van een SlideItem of het aanpassen van het level van de SlideItem heeft geen validatie om te controleren of het level wel mogelijk is. Dit zou opgelost worden door simpele validaties te doen op de waarden.

Er zijn geen tests aanwezig

Het programma bevat geen unit tests, dit is handig om te controleren of het programma nog steeds functioneert op de verwachte manier na nieuwe toevoegingen of aanpassingen gedaan worden.

Conventies

Er wordt geen rekening gehouden met consistentie of gebruik van ‘this’ keywoord.

Conclusie

JabberPoint is lang niet perfect en heeft veel tekortkomingen. Zo zijn er een aantal bugs die een nieuwe oplossing nodig hebben, namelijk het laden van niet bestaande afbeeldingen en dubbele sneltoetsen. Andere systemen zijn niet optimaal of schaalbaar gemaakt, deze kunnen opgelost worden door middel van een aantal Design Patterns toe te passen. Verder zijn er nog een aantal inconsistente kwaliteitsproblemen in de code zelf zoals methode namen, missende access modifiers en geen of slechte comments. Alle zaken in de eerder beschreven stukken worden aangeraden om aangepast te worden om de kwaliteit te verbeteren en zo het programma efficiënter en schaalbaar te maken.