Jabber Point Report

Ruike Yuan

This document describes the identified problem within Jabber point and applied solutions.

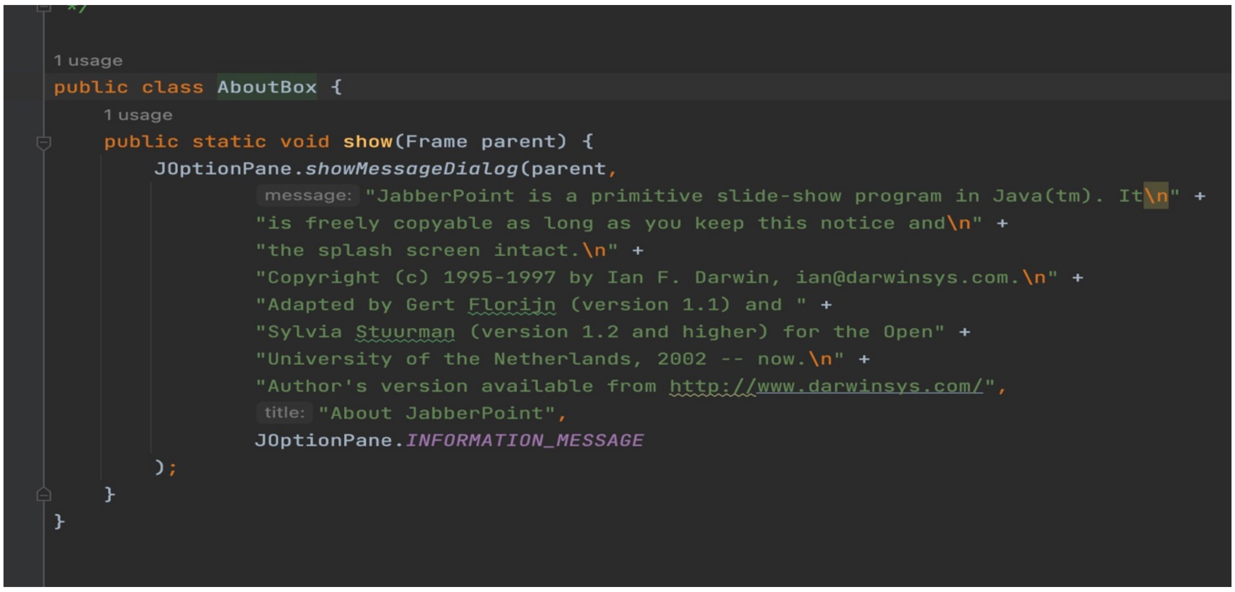
This document describes the identified problem within Jabber point and applied solutions.

Table of Contents

Menu controller:

The AboutBox class only does one functionality and the aboutBox should be shown when clicking the menu button “About”, so the method can be integrated to the ManuController class, which helps to maintain the code and provides clarity as less class needs to be checked and the menu button is directly linked to the method to be called (see pic. 1 and 2), making classes more internally conherent.

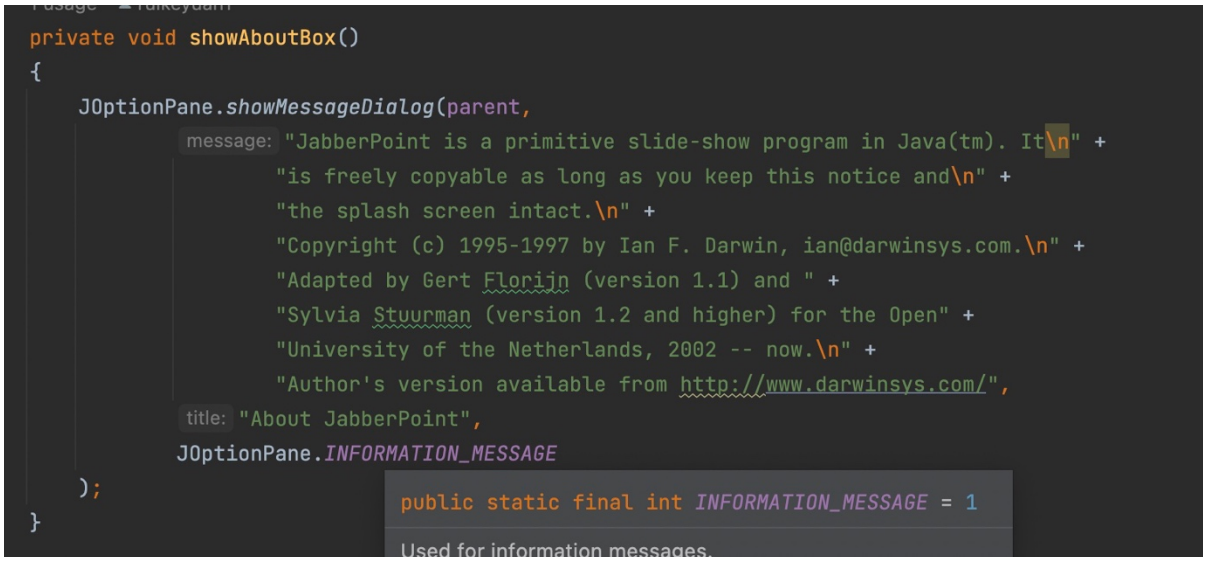
Before refactoring:



```
*/

1 usage
public class AboutBox {
    1 usage
    public static void show(Frame parent) {
        JOptionPane.showMessageDialog(parent,
                message: "JabberPoint is a primitive slide-show program in Java(tm). It\n" +
                "is freely copyable as long as you keep this notice and\n" +
                "the splash screen intact.\n" +
                "Copyright (c) 1995-1997 by Ian F. Darwin, ian@darwinsys.com.\n" +
                "Adapted by Gert Florijn (version 1.1) and " +
                "Sylvia Stuurman (version 1.2 and higher) for the Open" +
                "University of the Netherlands, 2002 -- now.\n" +
                "Author's version available from http://www.darwinsys.com/",
                title: "About JabberPoint",
                JOptionPane.INFORMATION_MESSAGE
        );
    }
}
```

Pic. 1

solution:

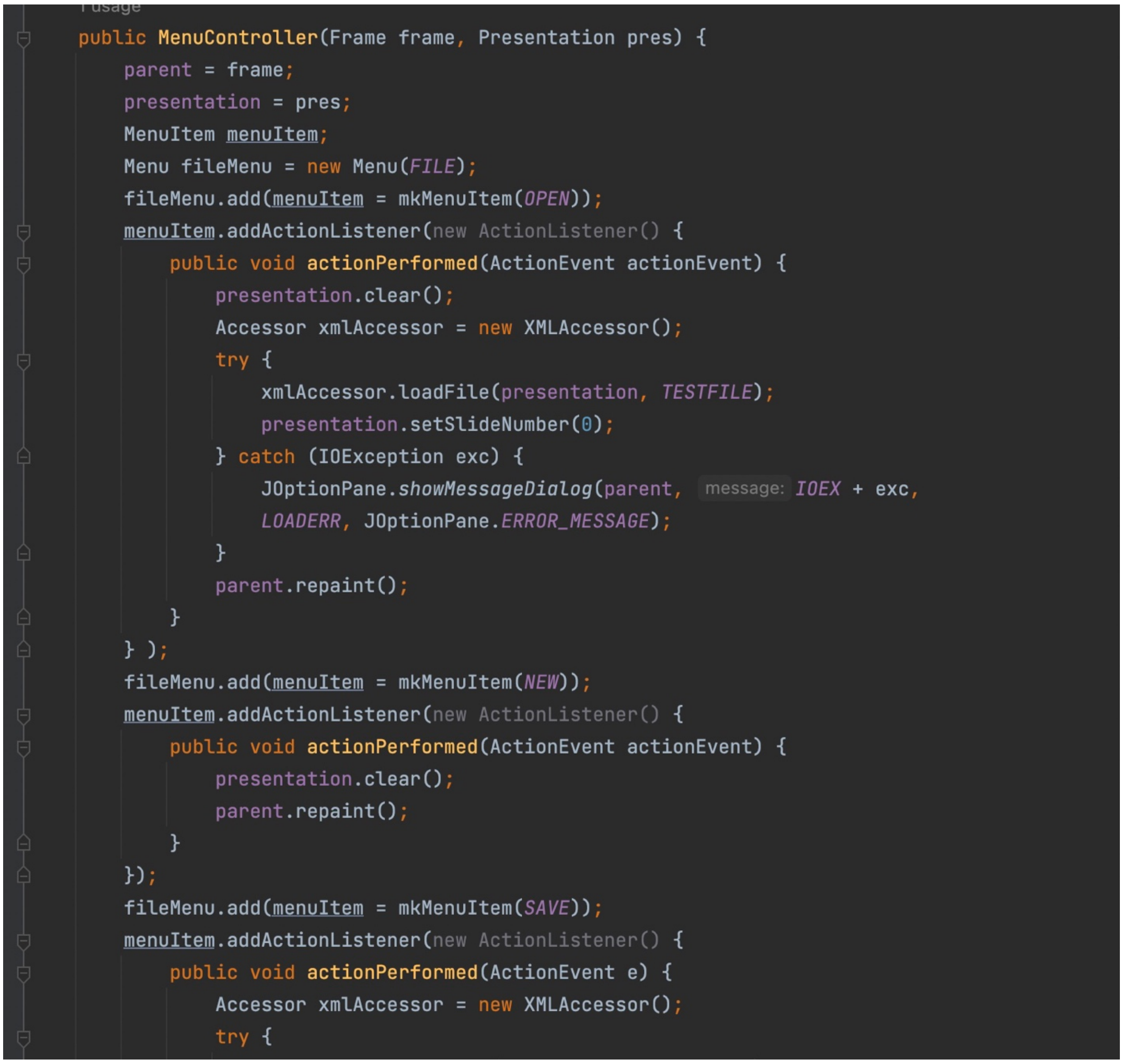


```java
private void showAboutBox()
{
    JOptionPane.showMessageDialog(parent,
            message: "JabberPoint is a primitive slide-show program in Java(tm). It\n" +
                    "is freely copyable as long as you keep this notice and\n" +
                    "the splash screen intact.\n" +
                    "Copyright (c) 1995-1997 by Ian F. Darwin, ian@darwinsys.com.\n" +
                    "Adapted by Gert Florijn (version 1.1) and " +
                    "Sylvia Stuurman (version 1.2 and higher) for the Open" +
                    "University of the Netherlands, 2002 -- now.\n" +
                    "Author's version available from http://www.darwinsys.com/",
            title: "About JabberPoint",
            JOptionPane.INFORMATION_MESSAGE
    );
}
```

```
public static final int INFORMATION_MESSAGE = 1
Used for information messages.
```

Pic. 2

Before refactoring:

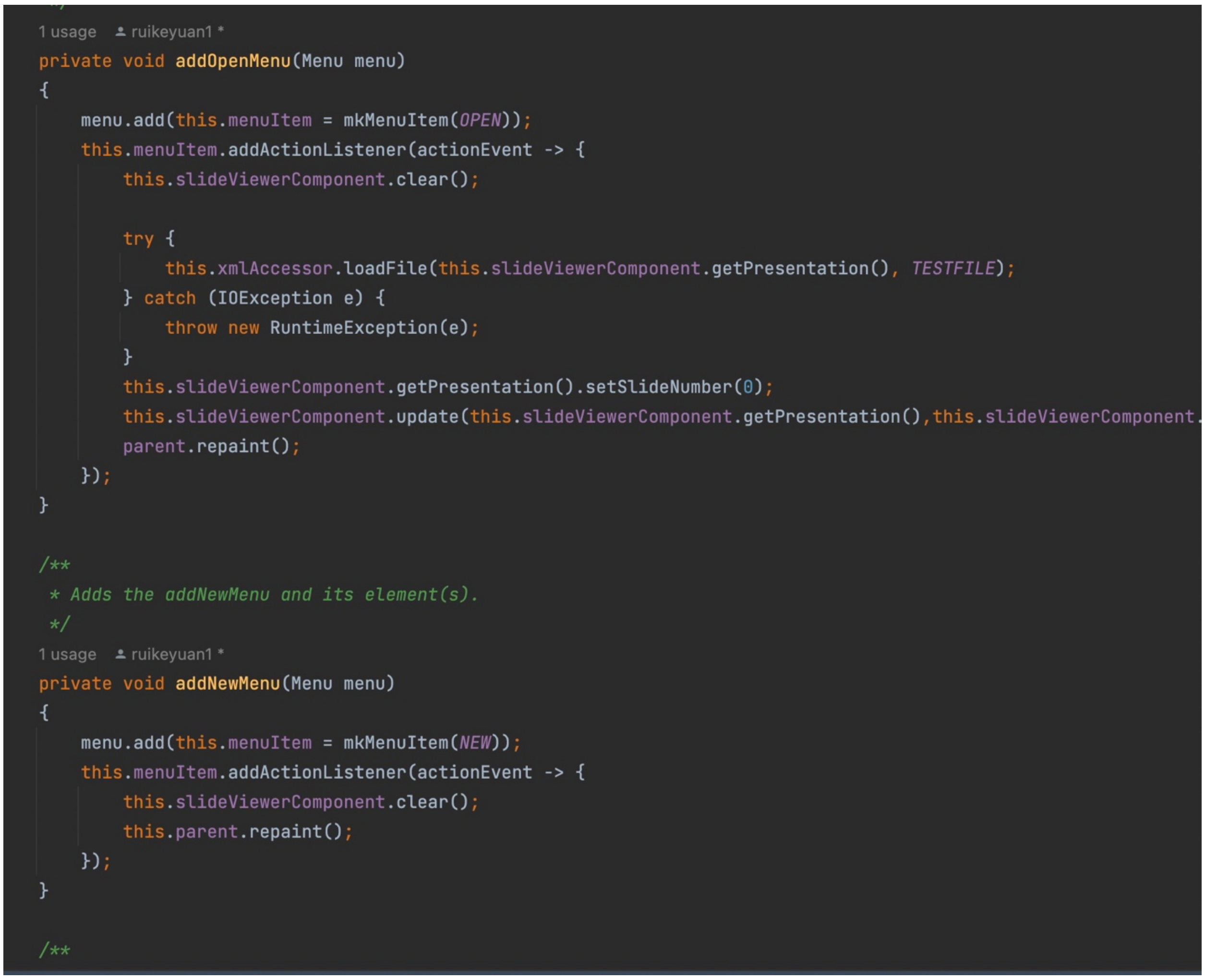


```
    public MenuController(Frame frame, Presentation pres) {
        parent = frame;
        presentation = pres;
        MenuItem menuItem;
        Menu fileMenu = new Menu(FILE);
        fileMenu.add(menuItem = mkMenuItem(OPEN));
        menuItem.addActionListener(new ActionListener() {
            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,  message: 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 {
```

Pic. 3

The contractor is too long, which violates the refactoring rule “long classes” and the page looks messy as all menu items their action listeners are added one after another. (see pic. 3)

Solution:



```
1 usage  ruikeyuan1 *
private void addOpenMenu(Menu menu)
{
    menu.add(this.menuItem = mkMenuItem(OPEN));
    this.menuItem.addActionListener(actionEvent -> {
        this.slideViewerComponent.clear();

        try {
            this.xmlAccessor.loadFile(this.slideViewerComponent.getPresentation(), TESTFILE);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
        this.slideViewerComponent.getPresentation().setSlideNumber(0);
        this.slideViewerComponent.update(this.slideViewerComponent.getPresentation(),this.slideViewerComponent.
        parent.repaint();
    });
}

/**
 * Adds the addNewMenu and its element(s).
 */
1 usage  ruikeyuan1 *
private void addNewMenu(Menu menu)
{
    menu.add(this.menuItem = mkMenuItem(NEW));
    this.menuItem.addActionListener(actionEvent -> {
        this.slideViewerComponent.clear();
        this.parent.repaint();
    });
}

/**
```

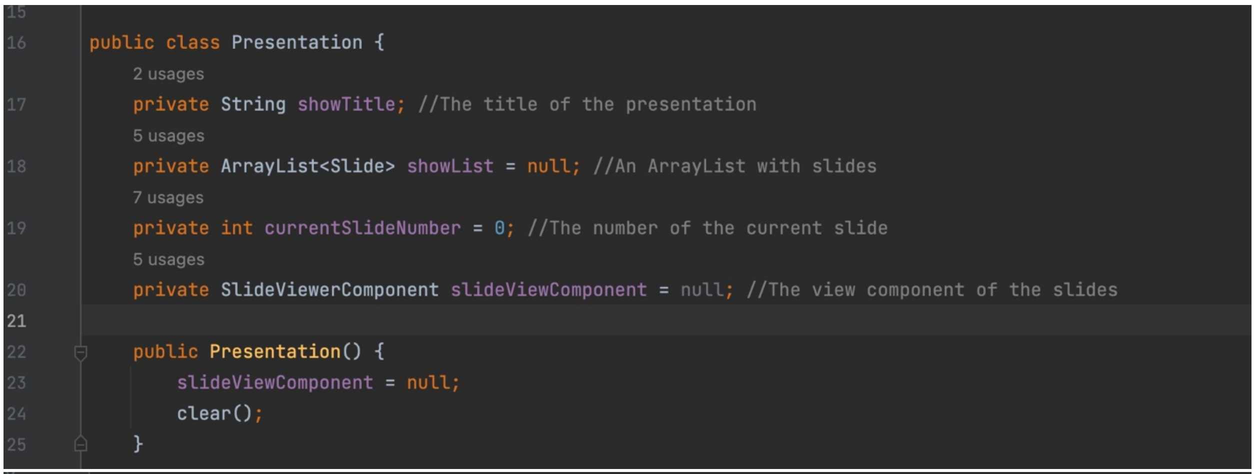
pic. 4

To easier to maintain the code, separate methods are created for adding each menu button with action listener within each function. This is fully adaptable and tested error free. (see pic. 4)

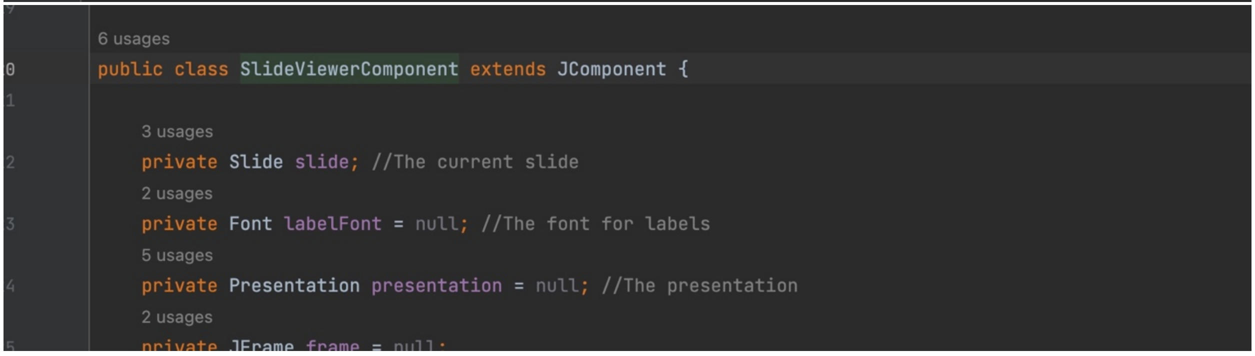
Remove Dual relationship

(between class Presentation and SlideViewComponent):

Before refactoring:



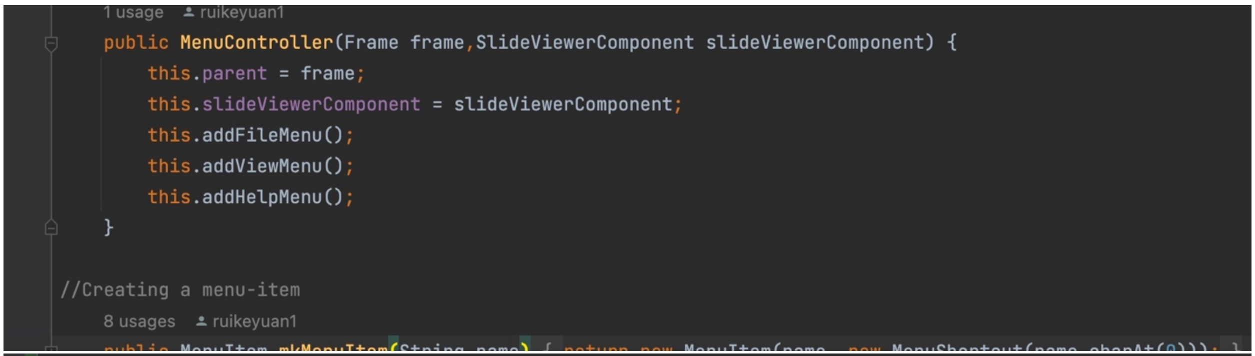
```
public class Presentation {
    2 usages
    private String showTitle; //The title of the presentation
    5 usages
    private ArrayList<Slide> showList = null; //An ArrayList with slides
    7 usages
    private int currentSlideNumber = 0; //The number of the current slide
    5 usages
    private SlideViewerComponent slideViewComponent = null; //The view component of the slides

    public Presentation() {
        slideViewComponent = null;
        clear();
    }
```



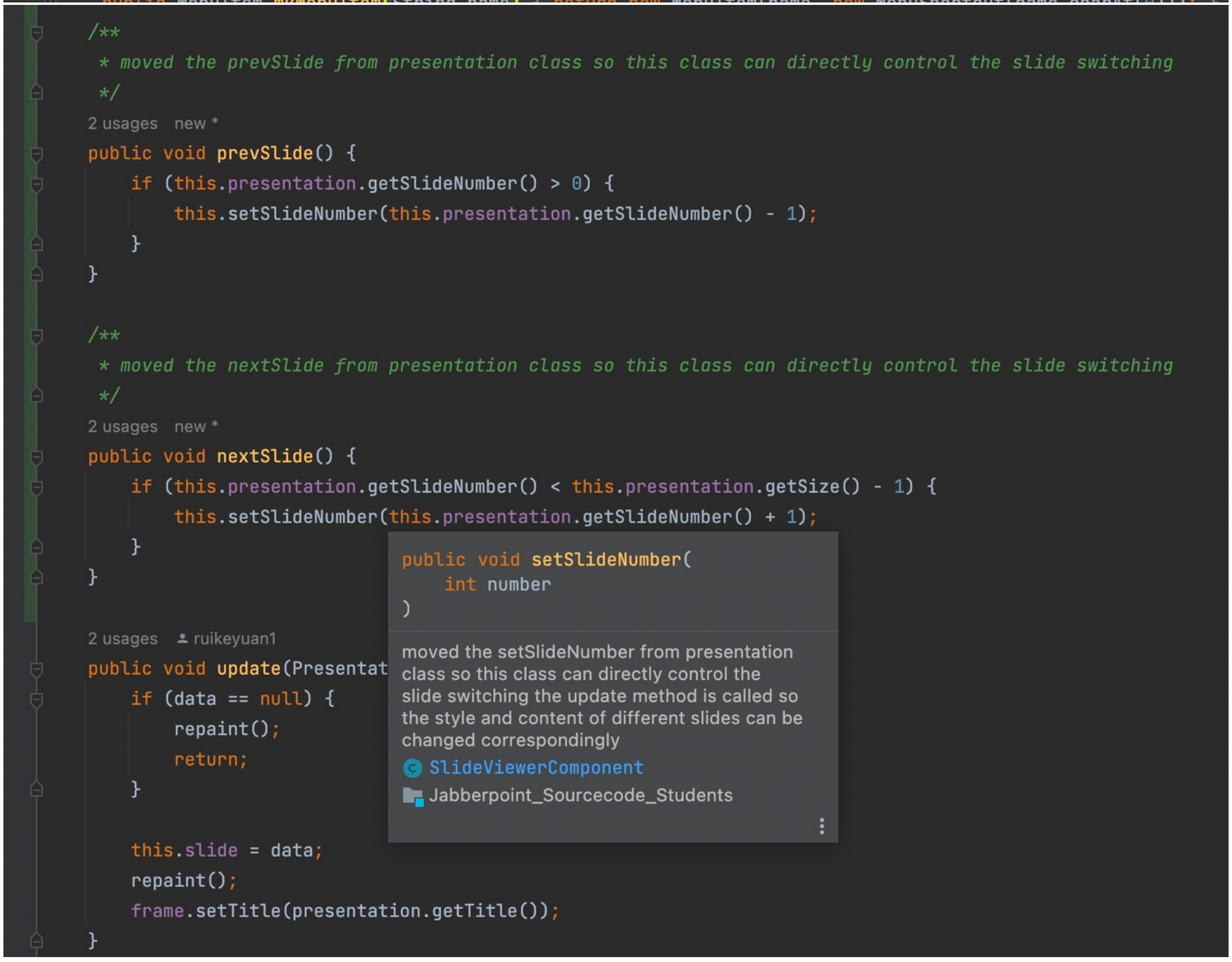
```
6 usages
public class SlideViewerComponent extends JComponent {

    3 usages
    private Slide slide; //The current slide
    2 usages
    private Font labelFont = null; //The font for labels
    5 usages
    private Presentation presentation = null; //The presentation
    2 usages
    private JFrame frame = null;
```

pic. 5 and pic.6

Two classes contain each other’s entity, which violates the rule of refactoring. To reduce the dependency between these two classes, several measures were taken.



```
1 usage   ruikeyuan1
public MenuController(Frame frame,SlideViewerComponent slideViewerComponent) {
    this.parent = frame;
    this.slideViewerComponent = slideViewerComponent;
    this.addFileMenu();
    this.addViewMenu();
    this.addHelpMenu();
}

//Creating a menu-item
   8 usages   ruikeyuan1
   public MenuItem mkMenuItem(String name) { return new MenuItem(name, new MenuShortcut(name.charAt(0))); }
```



```
/**
 * moved the prevSlide from presentation class so this class can directly control the slide switching
 */
2 usages   new *
public void prevSlide() {
    if (this.presentation.getSlideNumber() > 0) {
        this.setSlideNumber(this.presentation.getSlideNumber() - 1);
    }
}

/**
 * moved the nextSlide from presentation class so this class can directly control the slide switching
 */
2 usages   new *
public void nextSlide() {
    if (this.presentation.getSlideNumber() < this.presentation.getSize() - 1) {
        this.setSlideNumber(this.presentation.getSlideNumber() + 1);
    }
}

2 usages   ruikeyuan1
public void update(Presentat
    if (data == null) {
        repaint();
        return;
    }

    this.slide = data;
    repaint();
    frame.setTitle(presentation.getTitle());
}
```

```
public void setSlideNumber(
    int number
)
```

moved the setSlideNumber from presentation class so this class can directly control the slide switching the update method is called so the style and content of different slides can be changed correspondingly

SlideViewerComponent

Jabberpoint_Sourcecode_Students

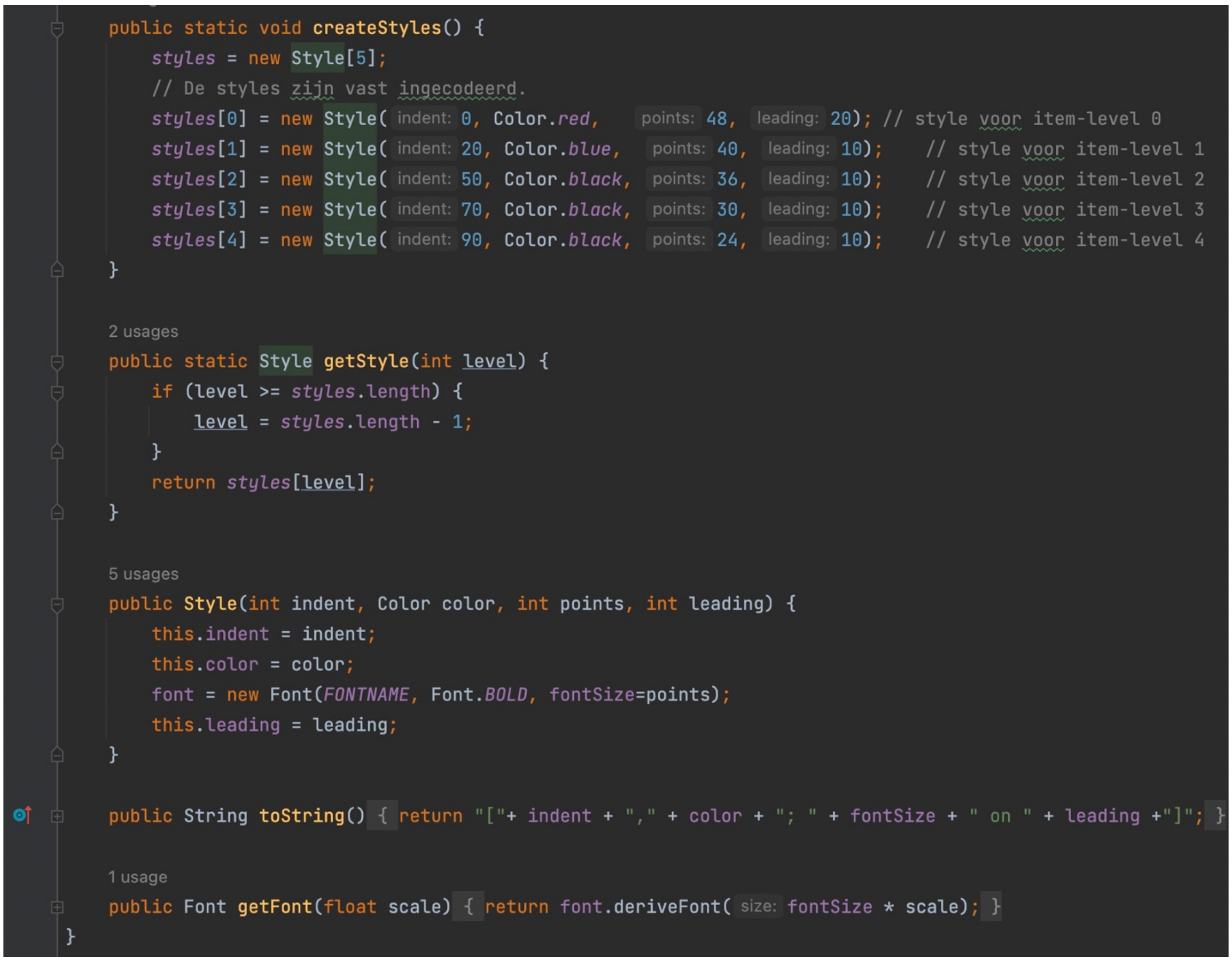
pic. 7 and pic.8

Main Steps:

1. Remove the field slideViewerComponent from Presentation Class
2. Add the field slide slideViewerComponent to keyControler and meanControler as these two class directly controls actions of presentation.
3. Move the methods in pic.8 from presentation class to slideViewerComponent so slideViewerComponents can directly controls the presentation.

Separate the style class:

Before refactoring:

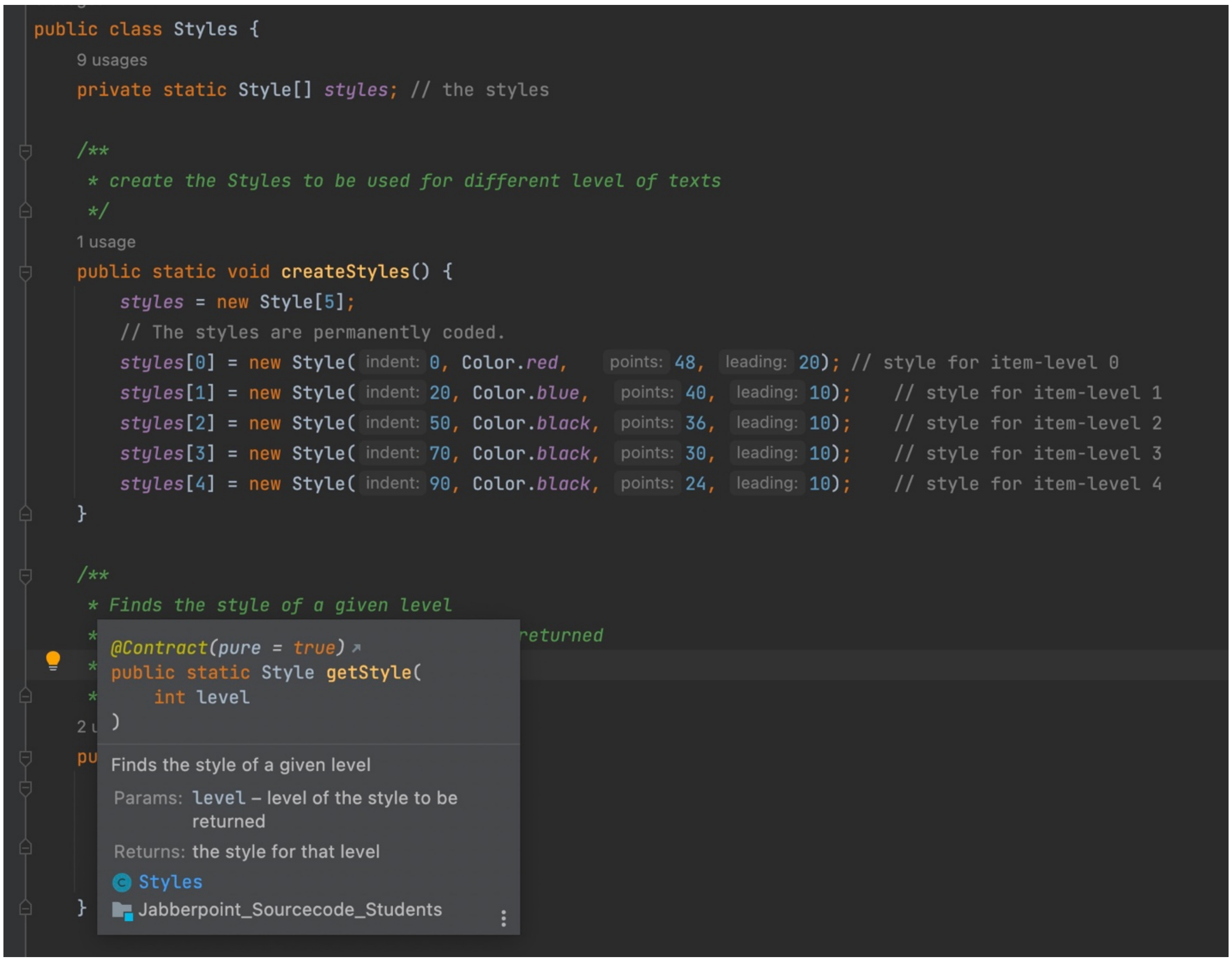


```java
    public static void createStyles() {
        styles = new Style[5];
        // De styles zijn vast ingecodeerd.
        styles[0] = new Style( indent: 0, Color.red,    points: 48,  leading: 20); // style voor item-level 0
        styles[1] = new Style( indent: 20, Color.blue,   points: 40,  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
    }

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

    5 usages
    public Style(int indent, Color color, int points, int leading) {
        this.indent = indent;
        this.color = color;
        font = new Font(FONTNAME, Font.BOLD, fontSize=points);
        this.leading = leading;
    }

    public String toString() { return "["+ indent + "," + color + "; " + fontSize + " on " + leading +"]"; }

    1 usage
    public Font getFont(float scale) { return font.deriveFont( size: fontSize * scale); }
}
```

Pic.9

The style class can be divided (for class hierarchical reason), the style class defines its private fields and methods but also the “createStyle”method that multiple objects of “Style” itself, which might make code viewer confused (see pic. 9)

Solution:



```
public class Styles {
    9 usages
    private static Style[] styles; // the styles

    /**
     * create the Styles to be used for different level of texts
     */
    1 usage
    public static void createStyles() {
        styles = new Style[5];
        // The styles are permanently coded.
        styles[0] = new Style( indent: 0, Color.red,    points: 48,  leading: 20); // style for item-level 0
        styles[1] = new Style( indent: 20, Color.blue,   points: 40,  leading: 10);   // style for item-level 1
        styles[2] = new Style( indent: 50, Color.black,  points: 36,  leading: 10);   // style for item-level 2
        styles[3] = new Style( indent: 70, Color.black,  points: 30,  leading: 10);   // style for item-level 3
        styles[4] = new Style( indent: 90, Color.black,  points: 24,  leading: 10);   // style for item-level 4
    }

    /**
     * Finds the style of a given level
     *                                      returned
     *
     *
    2 u
    pu



    }
```

```
@Contract(pure = true)
public static Style getStyle(
    int level
)
```

Finds the style of a given level

Params: level – level of the style to be returned

Returns: the style for that level

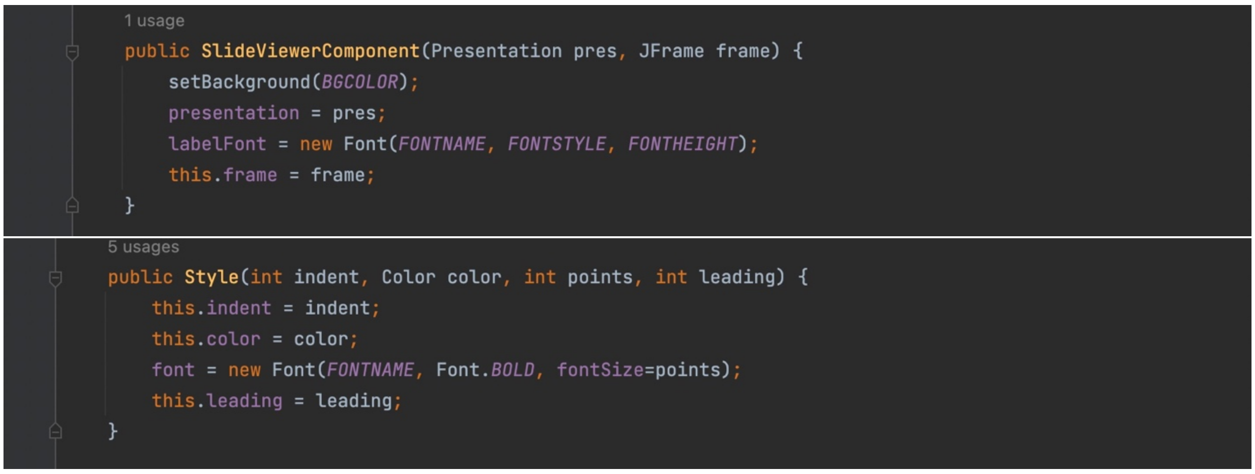
Styles

Jabberpoint_Sourcecode_Students

pic. 10

The solution is a new class “Styles” being created, with createStyles() getStyle() method moved to here and. Now this class contains the list of styles of the JabborPoint, making it clearer. (see pic.10)

Font Maintainer:

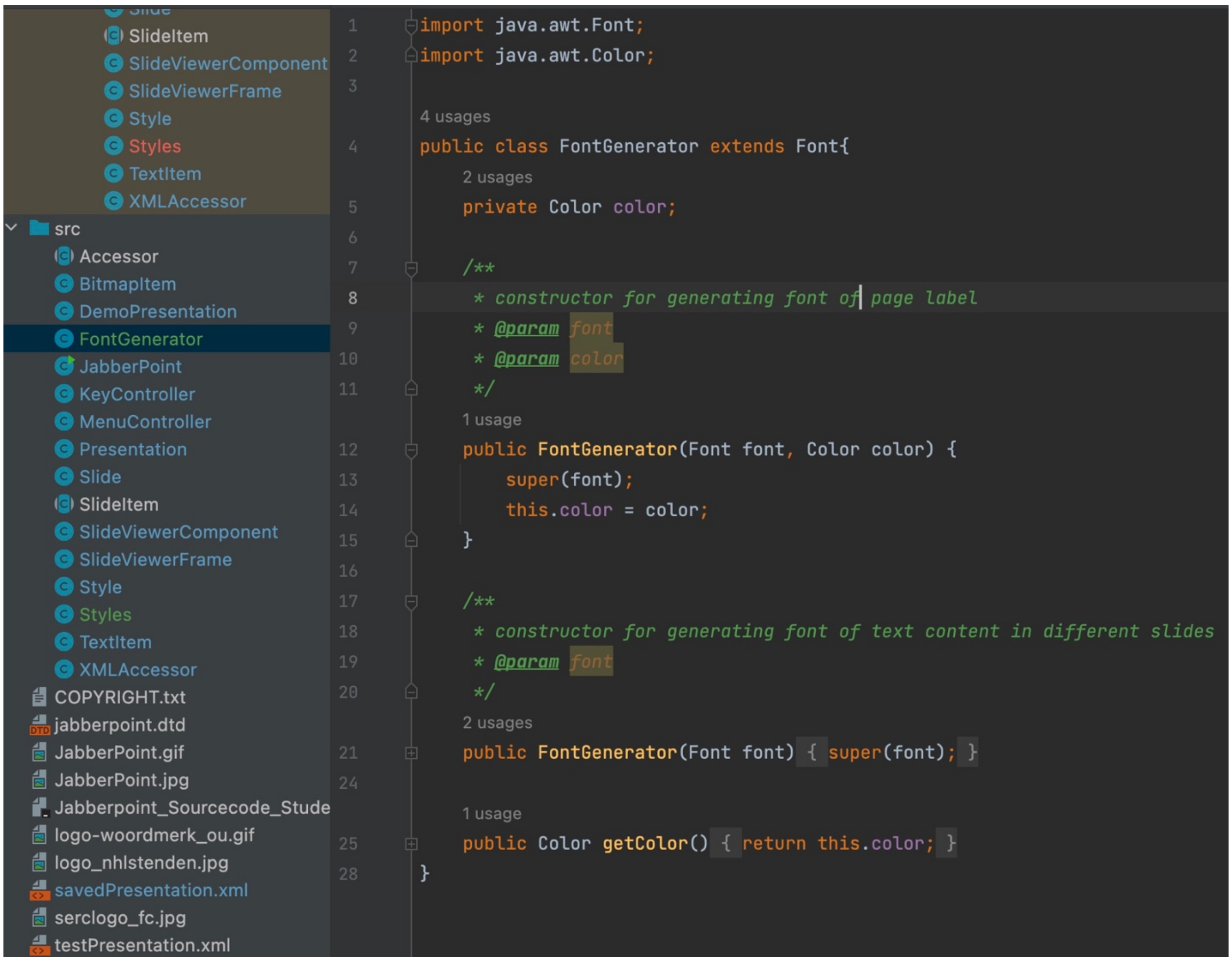


```
    1 usage
    public SlideViewerComponent(Presentation pres, JFrame frame) {
        setBackground(BGCOLOR);
        presentation = pres;
        labelFont = new Font(FONTNAME, FONTSTYLE, FONTHEIGHT);
        this.frame = frame;
    }
```

```
    5 usages
    public Style(int indent, Color color, int points, int leading) {
        this.indent = indent;
        this.color = color;
        font = new Font(FONTNAME, Font.BOLD, fontSize=points);
        this.leading = leading;
    }
```

pic. 11

There are two places in application where fonts (Font class) are defined and used, label font and text content respectively, but these two fonts were defined separately and I have an far-fetched idea to group them together in a class to provide more flexibility for future use.(see Pic.11)



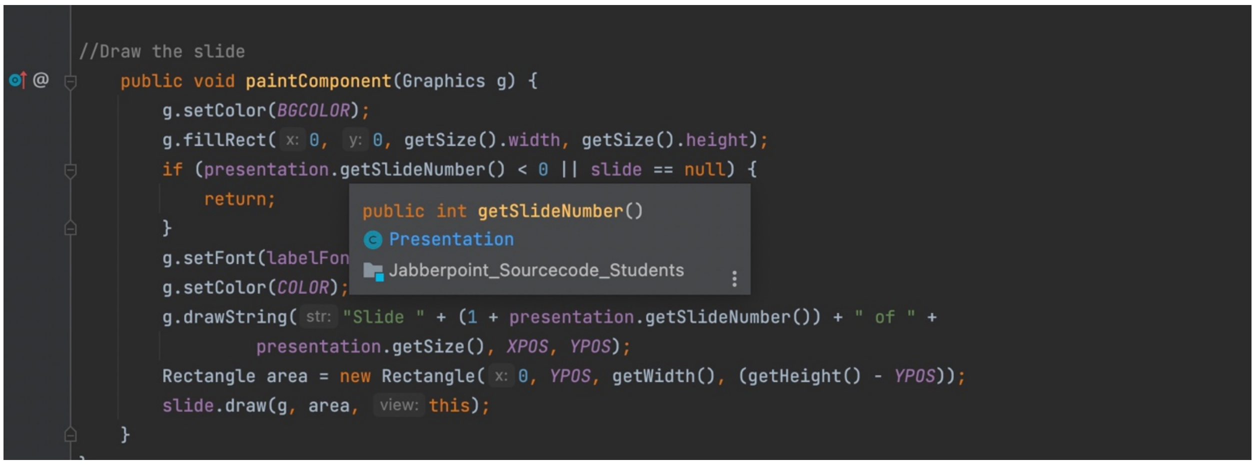
```java
import java.awt.Font;
import java.awt.Color;

4 usages
public class FontGenerator extends Font{
    2 usages
    private Color color;

    /**
     * constructor for generating font of page label
     * @param font
     * @param color
     */
    1 usage
    public FontGenerator(Font font, Color color) {
        super(font);
        this.color = color;
    }

    /**
     * constructor for generating font of text content in different slides
     * @param font
     */
    2 usages
    public FontGenerator(Font font) { super(font); }

    1 usage
    public Color getColor() { return this.color; }
}
```

Pic 12

A class “FontGenerator” were created which inherits the Font class of JAVA and have two different constructors for textContent and label respectively. So, when declaring a new font now, “fontGenerator” should be declared instead of Font (Java class) directly. (See pic.12)

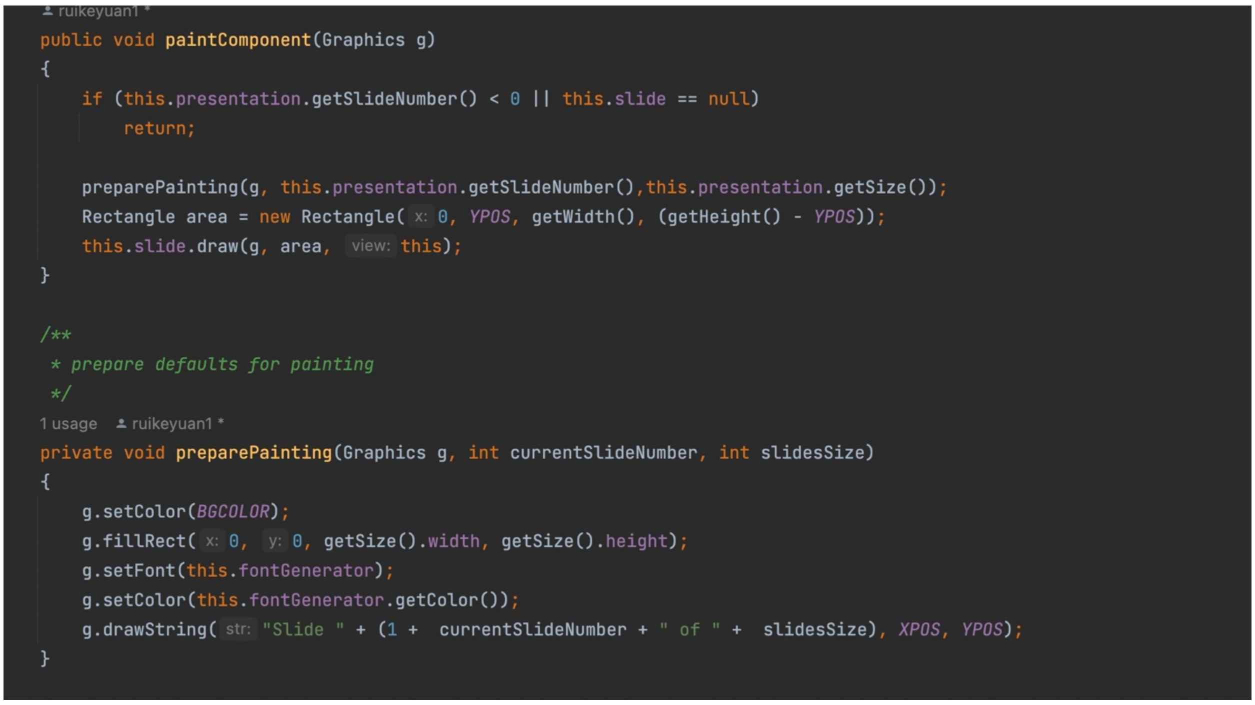
Class separation:

Before refactoring:



```java
    //Draw the slide
    public void paintComponent(Graphics g) {
        g.setColor(BGCOLOR);
        g.fillRect( x: 0,  y: 0, getSize().width, getSize().height);
        if (presentation.getSlideNumber() < 0 || slide == null) {
            return;
        }
        g.setFont(labelFon
        g.setColor(COLOR);
        g.drawString( str: "Slide " + (1 + presentation.getSlideNumber()) + " of " +
                presentation.getSize(), XPOS, YPOS);
        Rectangle area = new Rectangle( x: 0, YPOS, getWidth(), (getHeight() - YPOS));
        slide.draw(g, area,  view: this);
    }
```

pic.13

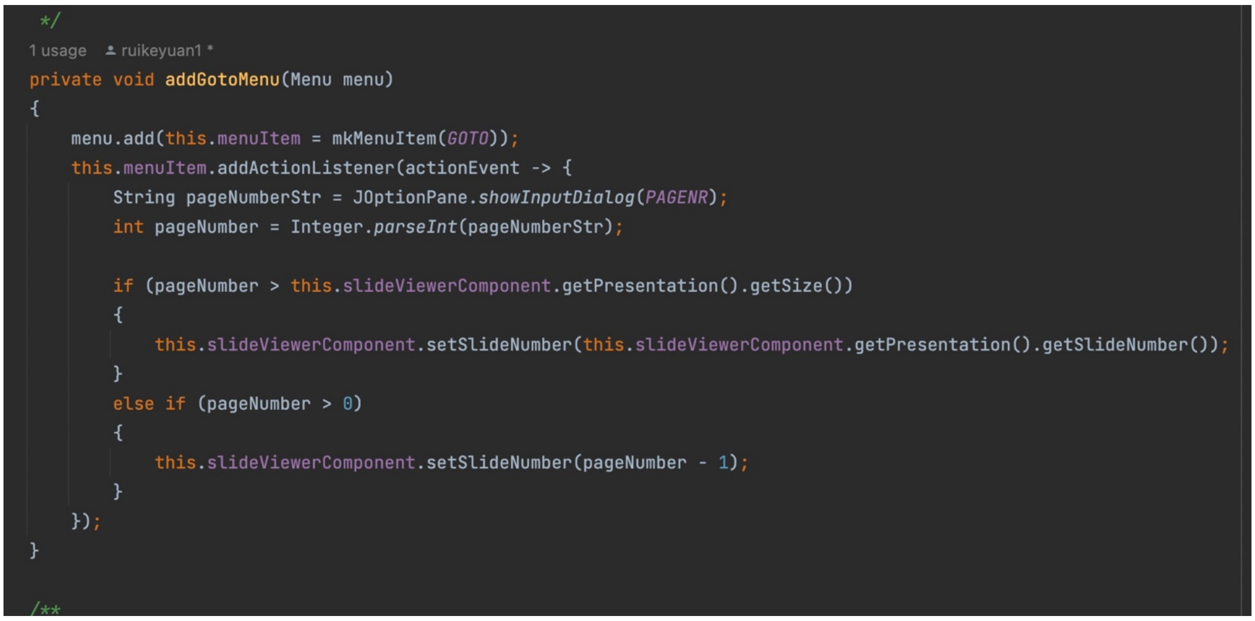


```
 ruikeyuan1 *
public void paintComponent(Graphics g)
{
    if (this.presentation.getSlideNumber() < 0 || this.slide == null)
        return;

    preparePainting(g, this.presentation.getSlideNumber(),this.presentation.getSize());
    Rectangle area = new Rectangle( x: 0, YPOS, getWidth(), (getHeight() - YPOS));
    this.slide.draw(g, area,  view: this);
}

/**
 * prepare defaults for painting
 */
1 usage   ruikeyuan1 *
private void preparePainting(Graphics g, int currentSlideNumber, int slidesSize)
{
    g.setColor(BGCOLOR);
    g.fillRect( x: 0,  y: 0, getSize().width, getSize().height);
    g.setFont(this.fontGenerator);
    g.setColor(this.fontGenerator.getColor());
    g.drawString( str: "Slide " + (1 +  currentSlideNumber + " of " +  slidesSize), XPOS, YPOS);
}
```

Pic.14

A class "paintComponent" can be separated for better readability and maintainability. The idea is to separate it into two methods, one for preparing the painting and one for taking actions (by calling the draw method) (see pic.13 and 14)

Error checking:



```
 */
1 usage   ruikeyuan1 *
private void addGotoMenu(Menu menu)
{
    menu.add(this.menuItem = mkMenuItem(GOTO));
    this.menuItem.addActionListener(actionEvent -> {
        String pageNumberStr = JOptionPane.showInputDialog(PAGENR);
        int pageNumber = Integer.parseInt(pageNumberStr);

        if (pageNumber > this.slideViewerComponent.getPresentation().getSize())
        {
            this.slideViewerComponent.setSlideNumber(this.slideViewerComponent.getPresentation().getSlideNumber());
        }
        else if (pageNumber > 0)
        {
            this.slideViewerComponent.setSlideNumber(pageNumber - 1);
        }
    });
}

/**
```

Pic.15

The "go to a specific page" functionality does not have a check method so there would be an error if user enters a value that is out of the boundaries of all slides. As a result, a check was added to make sure an error won't appear, and the page is redirects to either the first or last the page of the Jabber point accordingly. (see pic. 15)