System Analysis

Software Quality

Group 5

Cristian Trifan

Mihail Josan

This analysis covers the objectives, core components, design patterns, data flows, and non-functional considerations of the project.

NHL Stenden University of Applied Science, February 12th, 2025

Table of Contents

1. Objective and Scope

Purpose: JabberPoint is a primitive slide presentation tool written in Java. Although simple, it demonstrates key concepts in building a graphical, interactive presentation system.

Primary Functions:

Display a slide presentation using a graphical user interface (GUI).

Load presentation data from an XML file or use a built-in demo.

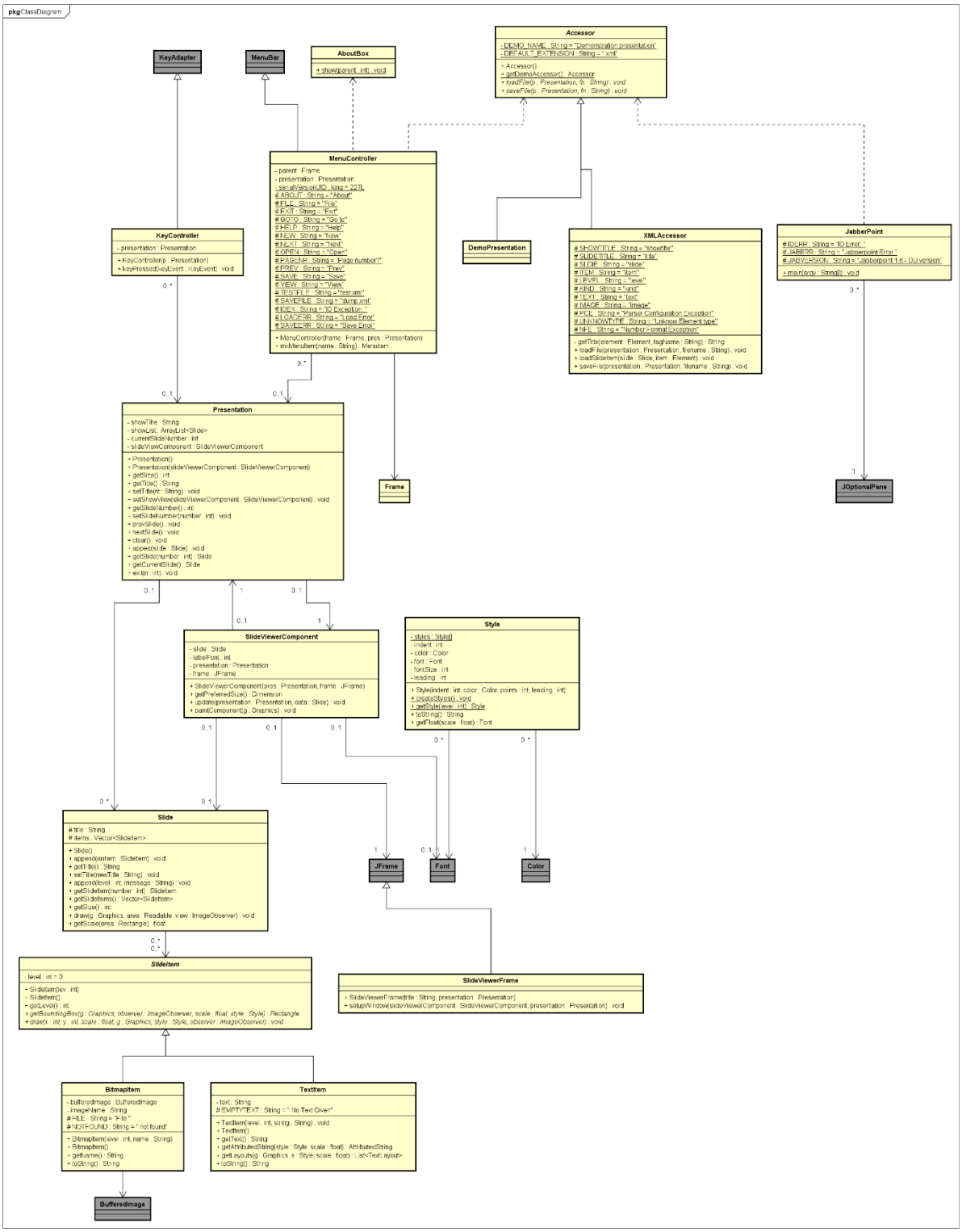
Render slide contents including text and images.

Allow user navigation (via keyboard and menu commands).

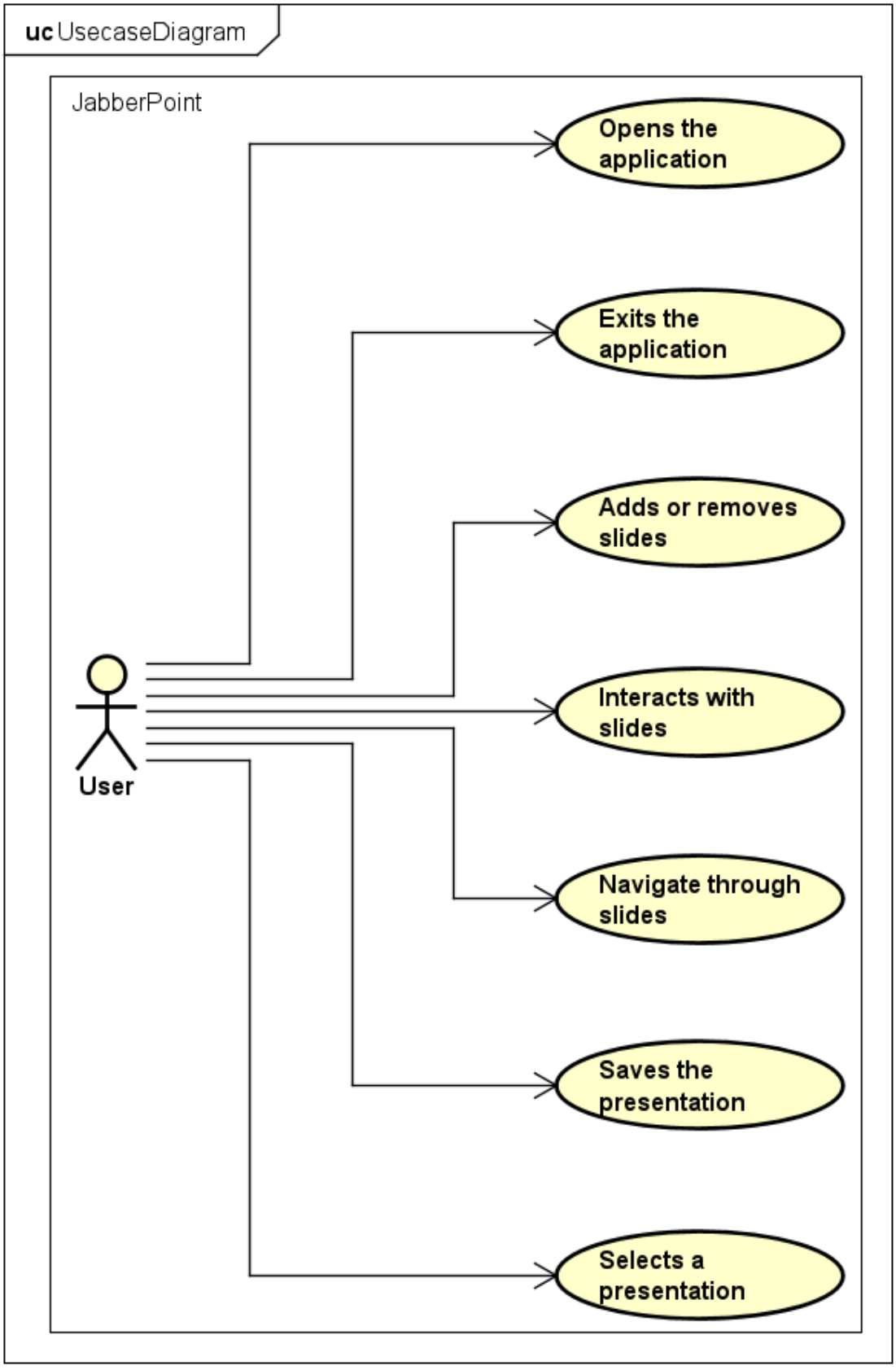
Support file operations like saving and opening presentations.

2. Diagrams

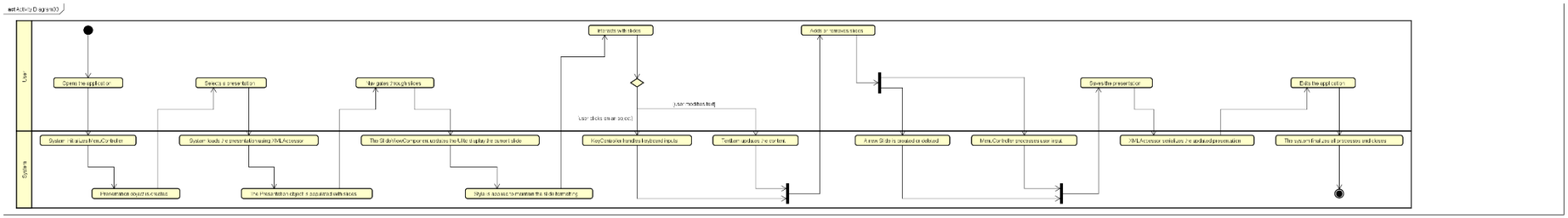
Class Diagram



UseCase Diagram



Activity Diagram



3. Architectural Overview

JabberPoint is designed following a simplified Model–View–Controller (MVC) architecture that promotes a clear separation of concerns:

Model: Maintains the presentation's data structures (slides, slide items, text content, images, style data). *Key Classes:* Presentation, Slide, SlideItem (and its subclasses TextItem and BitmapItem).

View: Provides the graphical user interface elements and rendering of slides. *Key Classes:* SlideViewerFrame, SlideViewerComponent, AboutBox.

Controller: Processes user input from the keyboard and menus and updates the model accordingly. *Key Classes:* KeyController, MenuController.

Data Access Layer: Abstracts reading from and writing to data files for presentations. *Key Classes:* Accessor (abstract), with concrete implementations such as XMLAccessor and DemoPresentation.

Utility/Support: Supports presentation styling and error handling. *Key Classes:* Style.

4. Detailed Component Breakdown

4.1 Presentation Model

Presentation:

Responsibilities:

Maintain an in-memory list of slides.

Handle navigation between slides (next, previous, jump-to).

Manage the current slide index.

Interactions:

Notifies the view (via SlideViewerComponent) when the current slide changes.

Slide:

Responsibilities:

Store the title and a collection of slide items.

Provide a draw() method that coordinates rendering of the slide title and it sitems.

Interactions:

Relies on each SlideItem to perform its own drawing and to report its size.

SlideItem Hierarchy:

Abstract Class (SlideItem):

Defines the interface for drawing and measuring item bounds.

Concrete Classes:

TextItem: Renders formatted text using Java's AWT text layout facilities.

BitmapItem: Loads and renders an image from the file system.

Interactions:

Both types use the Style class to determine visual formatting (indentation, font, color).

4.2 User Interface (View)

SlideViewerFrame:

Responsibilities:

Create the main application window.

Instantiate and display the SlideViewerComponent.

Set up the window’s size, key listeners, and menu bar.

Interactions:

Incorporates controllers (keyboard and menu) to connect user actions with model updates.

SlideViewerComponent:

Responsibilities:

Render the current slide.

Provide feedback (e.g., current slide number) on screen.

Interactions:

Calls the draw() method of Slide and its items.

Receives updates from the Presentation model when navigation occurs.

AboutBox:

Responsibilities:

Display application information (copyright, version).

Interactions:

Triggered via the "About" menu command from MenuController.

4.3 Controllers

KeyController:

Responsibilities:

Listen for key events (such as PgUp, PgDn, arrow keys, Enter, etc.).

Map key events into navigation commands (next slide, previous slide, exit).

Interactions:

Directly updates the Presentation model, triggering a view refresh.

MenuController:

Responsibilities:

Construct the menu bar with various commands (File, View, Help).

Manage actions for operations like opening a file, starting a new presentation, saving, and navigation.

Interactions:

Invokes methods on the Presentation model or uses an Accessor (e.g., XMLAccessor) to deal with file I/O.

4.4 Data Access Layer

Accessor (Abstract) and its Implementations:

Responsibilities:

Define the interface for loading and saving presentation files.

Concrete Implementations:

XMLAccessor:

Reads an XML file using a DOM parser.

Iterates through XML nodes to create Slide and SlideItem objects.

Writes an XML representation when saving.

DemoPresentation:

Provides a hard-coded demonstration presentation when no file is provided.

Interactions:

The main entry point in JabberPoint decides which implementation to invoke based on startup parameters.

4.5 Styling

Style:

Responsibilities:

Define visual properties (indentation, color, font size, leading) for slide items.

Maintain an array of preset styles corresponding to different item levels (hierarchical rendering).

Interactions:

Accessed by TextItem and others to ensure consistency in presentation.

5. Data Flow and Interactions

1. **Startup:**

   The application is launched from JabberPoint.main().

   The system calls Style.createStyles() to initialize presentation styles.

   A Presentation object is created along with a SlideViewerFrame that displays the SlideViewerComponent.

   Depending on command-line arguments, either a demo presentation (DemoPresentation) or an XML file (handled by XMLAccessor) is loaded into the Presentation model.

2. **User Navigation:**

   Keyboard:

   Events captured by KeyController update the current slide number.

Menu:

Menu selections (via MenuController) trigger file operations (e.g., open, save) and navigation commands.

Upon any update, the Presentation model calls update() on the SlideViewerComponent to trigger a re-render.

3. **Rendering:**

   The SlideViewerComponent handles draw requests.

   It delegates the drawing of each slide to the corresponding Slide object, which in turn calls the draw() method of each SlideItem.

   Styling and scaling factors are applied during the drawing process for proper display.

4. **File I/O:**

   When saving, the presentation's state is written to an XML file using XMLAccessor.

   During load, the XML structure is parsed to recreate each slide and its items, rehydrating the presentation model.

6. Non-functional Aspects and Considerations

Usability:

Simple, minimalistic user interface.

Keyboard shortcuts and menu operations provide basic accessibility.

Extensibility and Maintainability:

The use of an abstract Accessor allows future extension to support other file formats or improved XML handling.

The MVC-like separation of model, view, and controller components allows for easier maintenance.

Performance:

The project is designed to handle small to moderate presentations.

XML parsing using DOM is sufficient for modest data sizes but might present scaling issues with very large presentations.

Technology Stack:

Built using Java's AWT and Swing libraries, technologies that are mature but might be considered dated for modern UI needs.

Error Handling:

Basic error handling is implemented (e.g., catching I/O exceptions and displaying error dialogs via Swing).

Some error feedback (such as logging errors via System.err) could be improved for a production-quality application.

7. Errors

1. Inconsistent Comments Throughout the code, there are inconsistent comments that make the code harder to understand. It's important to maintain clarity by standardizing comments across the project.

2. Missing Access Modifiers Some parts of the code are missing necessary access modifiers. These should be added to ensure proper encapsulation and visibility control.

3. Use of this Keyword In several constructors and methods, the this. keyword is missing. For example, in:

- KeyController.java (lines 18, 27, 32)
- MenuController.java (lines 49, 50, 56, 59, 60, 62, 65, 71, 72, 80, 82, 91, 99, 105, 113, 121)
- Presentation.java (lines 23, 33, 37, 41, 50, 55, 56, 57, 63, 64, 70, 71, 77, 91, 96)
- SlideViewerComponent.java
- Slide.java
- Style.java

Adding this. will help avoid confusion and improve clarity.

4. Incorrect Image Reference In DemoPresentation.java (line 47), there is a wrong reference to an image. This needs to be corrected to ensure the proper functionality of the code.

5. Vague Parameter Names In several methods, vague parameter names are used. It is important to use more descriptive and meaningful names to improve code readability. This applies to:

- Accessor.java (loadFile() and saveFile() methods)
- SlideItem.java (getBoundingBox() and draw() methods)
- BitmapItem.java (getBoundingBox() and draw() methods)
- TextItem.java (getBoundingBox() and draw() methods)
- KeyController.java (constructor)
- MenuController.java (constructor)
- Presentation.java (setTitle() method)
- SlideViewerComponent.java (constructor and paintComponent() method)

- Slide.java (draw() method)

6. Unnecessary Casts Unnecessary type casts appear in:

- SlideItem.java
- Slide.java
- XMLAccessor.java
- Presentation.java (line 91)

These casts should be removed to streamline the code and reduce potential confusion.

7. Hardcoded Styles Style.java contains hardcoded styles in the createStyles() method. To improve flexibility and maintainability, styles should be externalized or passed as parameters where necessary.

8. Random Whitespace

- XMLAccessor.java (line 52) contains an unnecessary blank line that should be removed.

9. Inconsistent Indentation There is inconsistent indentation in the following files:

- BitmapItem.java
- SlideItem.java
- TextItem.java
- MenuController.java (line 127)
- Slide.java (line 65)
- SlideViewerComponent.java (line 59)
- SlideViewerFrame.java (line 31)

Consistent indentation should be applied throughout the code to improve readability and maintain a uniform style.

10. Unused Constants There are unused constants in:

- Accessor.java
- XMLAccessor.java

These constants should either be removed or properly used to avoid unnecessary code.

11. Useless Field Assignment

- In SlideItem.java, there is an unnecessary assignment of a value to the level field. This should be addressed to clean up the code.

12. Absence of Constructors

- XMLAccessor.java lacks a constructor, which can lead to potential issues when initializing objects of this class. A constructor should be implemented to ensure proper initialization.

13. Absence of Unit Tests The code lacks (unit) tests. Proper testing should be implemented to ensure the code functions as expected and to avoid introducing bugs in future updates.

By addressing these errors, the code quality can be greatly improved, making it more maintainable and understandable.

8. Summary and Future Recommendations

Summary:

JabberPoint is a well-structured, educational example that demonstrates how to build a presentation tool with a clear separation between data, presentation, and control logic. Its layered architecture makes it a good example for understanding MVC concepts.

Future Enhancements:

UI Improvements:

Consider migrating to more modern GUI frameworks (such as JavaFX) for improved look and feel.

Enhanced Error Handling:

Improve robustness by refining error messages and recovery strategies.

Scalability:

Switch to a more efficient XML parsing method (like SAX or StAX) if the presentation data sets grow large.

Dynamic Styling:

Allow user configuration of styles instead of using hard-coded values.