JABBER POINT

Report

Ameli Fernando

Table of Contents

Challenges in General Design

This chapter explores common design challenges encountered in the development of JabberPoint.

Code Bloat:

The presentation component was overloaded with responsibilities primarily due to its large classes. A class should ideally be limited to its specific responsibilities and functions. The solution implemented involved breaking down these large classes by introducing new ones and extracting subclasses and interfaces where necessary.

This approach also addresses issues of code duplication, enhancing code clarity and improving readability. For instance, this strategy was applied effectively to the "Slide Viewer Component" and "Menu Controller," which were decomposed into several distinct classes with defined functionalities.

Data Clumps:

JabberPoint encountered the issue of data clumps, where certain classes contained excessive variables and signs. To address this, separate classes were created, and some fields were relocated to these new classes. This restructuring not only made the code more understandable but also enhanced its overall organization. This solution was implemented in classes such as the "Slide Viewer Component," "Menu Controller," "Slide," and "XMLAccessor."

Object-Oriented Misuse:

The presentation control in JabberPoint was flawed, with issues in the extension of communication and an imbalanced hierarchy. The resolution involved extracting new classes, allocating the appropriate functions to these classes, and shifting away from reliance on inheritance. This approach was applied specifically to the “XMLAccessor” and “Demo Presentation” classes. As a result, the organization and clarity of the code significantly improved.

Complexity in Classes:

Several classes within JabberPoint were overly complex statements. The adopted solution was to simplify these complexities by extracting methods, replacing type code with state, and converting complex switch statements into simpler if, while, and for loops. This method was specifically applied to the “Slide” and “TextItem” classes. As a result, the organization and clarity of the code were significantly increased.

Change Triggers:

JabberPoint had issues with unrelated methods requiring modifications whenever changes were made to a class. The solution involved splitting some classes and extracting subclasses to better isolate functionalities. This strategy was particularly effective with the “XMLAccessor” and “Demo Presentation” classes, as well as the “Loader” and “Saver” interfaces. As a result, the organization, clarity, and readability of the code improved, and code duplication was significantly reduced.

Redundancies:
It has been noted that code duplication is a frequent issue that affects many classes. The approach taken to address this involved creating new methods and classes and modifying existing statements and algorithms. These changes simplified the code, making it easier to understand and read.

Interdependencies:
An issue with incomplete library integration was identified in certain classes, including the "About Box." The resolution involved the removal and addition of various libraries to ensure the classes functioned correctly.

Minor Adjustments for Subtle Issues
This chapter explores small issues identified in the code and the subtle changes made to address them:

- Constants across nearly all classes were renamed to be more descriptive and were formatted in uppercase to align with Java conventions.
- In the “AboutBox” class, the issue of hardcoded strings was addressed. The solution involved restructuring the code into three methods: “show” and “showAboutBox,” both without return types, and “readAboutBoxMessageFromFile.” This last method facilitates reading the About Box message from a file, thus allowing for easier updates without direct code modifications. The implementation of “readAboutBoxMessageFromFile” leverages the “java.io.BufferedReader,” “java.io.FileReader,” and “java.io.File” classes to efficiently read the message. This method processes each line from the "about.txt" file, located in the same directory as the code, concatenates these into a single string, and returns the string.

The following classes were added to the imports to support this functionality:
```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
```

Class Modifications
This chapter will outline specific changes made to various classes.

DemoPresentation Class
The DemoPresentation class has been updated through several targeted refactorings to streamline its operations and improve code clarity. Here are the specific changes implemented:

1. **Import Optimization**: Essential classes, such as **Arrays**, have been imported to streamline array management within the class.
2. **Code Deduplication**: Repeated code segments, specifically those involving **slide.append**, have been replaced with an array-based approach that organizes slide information more efficiently, enhancing both readability and maintainability.

KeyController Class
The KeyController class has undergone specific refactoring to optimize its functionality and code structure. Here is the key change implemented:

1. **Replacing Switch/Case with HashMap**: To address code duplication and simplify condition handling, the previous switch/case structure has been replaced with a HashMap. This modification maps keys directly to their associated actions, streamlining the code and improving execution efficiency.
2. **Maintainability**: With commands and their associated keys clearly separated from the event handling logic, the code becomes easier to maintain and understand.

3. **Extensibility**: The use of HashMaps makes it easier to add new key bindings or modify existing ones, enhancing the class's extensibility.
4. **Error Handling**: Adding error handling within the **keyPressed** method could further enhance robustness, ensuring that unbound keys do not lead to unexpected behaviors or errors.

BitmapItem Class
The BitmapItem class has undergone several refinements to improve code quality and readability. Here are the specific changes made:

1. **String Formatting**: String constants were consolidated into a single constant, enhancing readability.
2. **Error Output**: String constants within **System.err.println** statements were replaced with formatted strings to streamline the output process.
3. **Parameter Renaming**: Parameters in the BitmapItem method were renamed for clarity—from **int level** and **String name** to **itemLevel** and **filename** respectively.
4. **Method Renaming**: The **draw** method was renamed to **drawImage** to more accurately describe its functionality.
5. **Improvements to String Method**: The **toString** method was annotated with **@Override** to clarify its role as an overridden method. Additionally, the formatting of the **toString** method was improved to enhance code readability.

Presentation class
The Presentation class currently integrates most of the other classes and nearly all elements of the application. A practical solution to this issue would be to segment it into several distinct classes, including the “Slide Viewer Component,” “Key Controller,” “Slide Viewer Frame,” “Menu Controller,” and “Slide.” This division would allow for the delegation of various functions, enhancing the structure of the system and simplifying the code.

Class Style
The Style class has a series of refinements to align with best practices in Java coding and to improve the overall code structure. Below are the specific enhancements made:

1. **Naming Conventions**: The constant variable **FONTNAME** was renamed to **FONT_NAME** to adhere to Java's naming conventions, which recommend using underscores to separate words in constant names. This change improves the readability of the code.
2. **Access Modifiers**: The access level of class variables was modified by adding the **private** modifier. This change ensures that class fields are encapsulated properly, restricting direct access from outside the class. Encapsulation is a fundamental principle of object-oriented programming that helps in maintaining the integrity of the data within the objects.

SlideViewerFrame Class
The SlideViewerFrame class has received updates aimed at improving clarity, maintainability, and compliance with Java coding best practices. These changes not only streamline the code and improve its organization but also ensure that the class adheres more closely to object-oriented design principles. This makes the SlideViewerFrame class more robust, easier to manage, and less prone to errors.

Here are the improvements made:

1. **Renaming Constants**: The constant **JABTITLE** was renamed to **JABERPOINT_TITLE** to better reflect its purpose and ensure the name is self-explanatory. This aligns with Java best practices for naming constants, making the code more readable and understandable.
2. **Reordering Modifiers**: The declaration sequence of the constants **WIDTH** and **HEIGHT** was changed from "public static final" to "public final static." Although both orders are technically correct in Java, the convention typically follows the "public static final" order. This refactoring might be an oversight, as the standard practice recommended by the Java Language Specification is to use "public static final."

3. **Repositioning Fields**: The fields **presentation** and **slideViewerComponent** were moved to the top of the class structure. This adjustment makes these fields more prominent and logical since they are critical in the **setupWindow** method, from which they were removed to reduce redundancy and improve clarity.
4. **Method Visibility Adjustment**: The **setupWindow** method was changed to private because it is only utilized within the SlideViewerFrame class. This change encapsulates the method, preventing external access and modifying it only when necessary from within the class, thus enhancing the security and integrity of the class's internal structure.