Design Document

For

Authorizing Application for Braille-Based Device

Prepared by:
Dilshad Khatri
Drew Noel
Jonathan Tung
Alvis Koshy

Prepared on: April 5, 2017
For: EECS 2311 – Software Development Project

Table of Contents

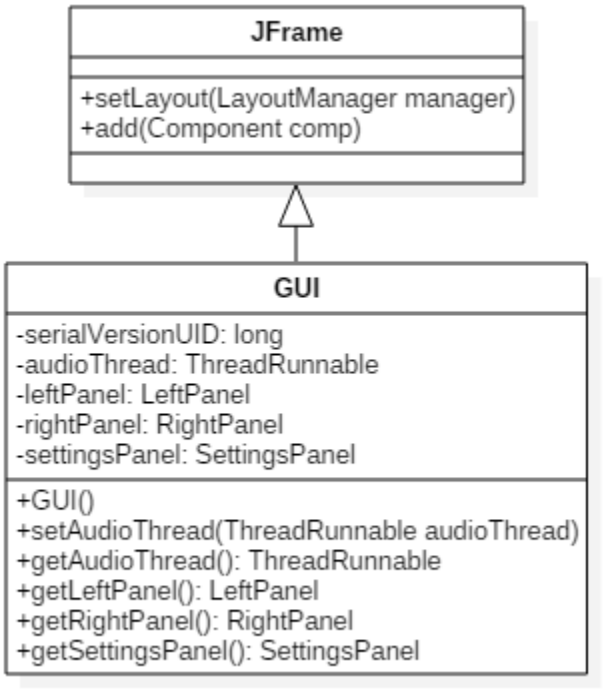
1. Introduction

This design document will discuss about the organization and structure of the player authorizing app. It will discuss high level organization of system as well as individual components by walking though important classes and methods and how they interact with each other. It will discuss the class diagram as well as sequence diagram which clearly illustrates the structure of Authorizing App.

2. Class Diagram

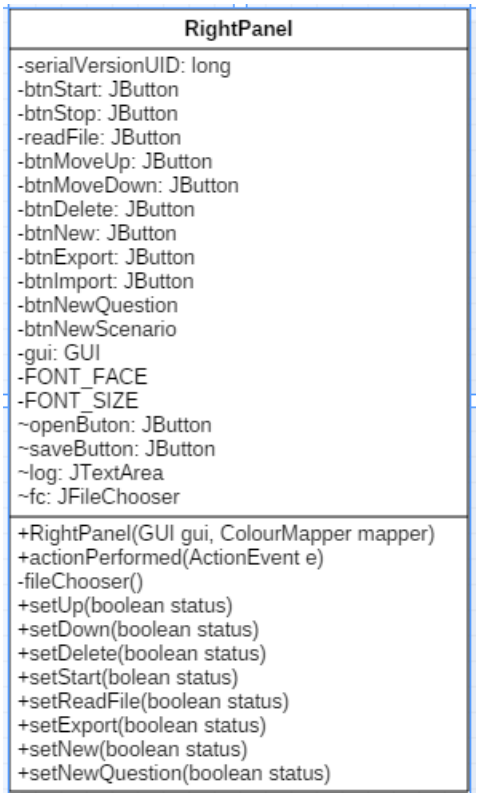
Class diagram of Authorizing App starts in common package at EntryPoint class which starts thread of MainThread class as shown in authorizingAppClassDiagram.jpeg. MainThread class simply sets up JFrame and initializes the App.

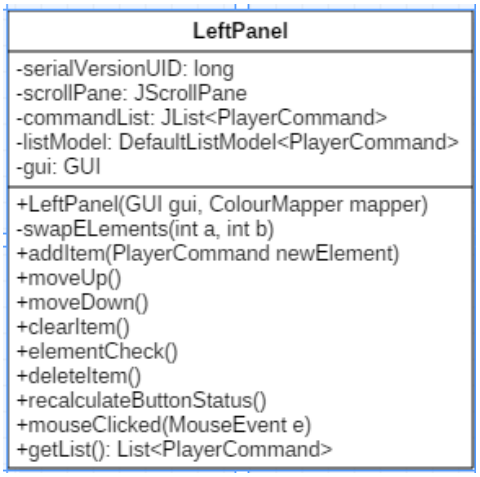
One of the most important class is GUI from which all classes hold composition relationship with. Life cycle of all classes depend on GUI class since it is the skeleton for Authorizing App. There are four main classes that come under GUI class, LeftPanel.java, RightPanel.java, SettingsPanel.java, ThreadRunnable.java. These classes can be accessed by the methods shown in GUI UML diagram below.

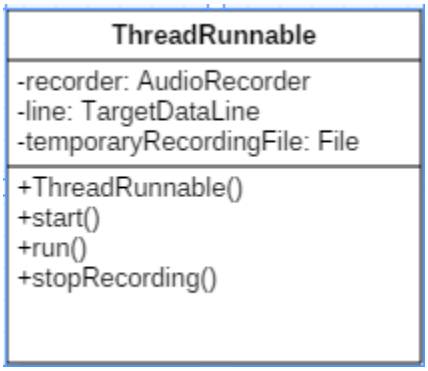


Going back to the class diagram image(jpeg), LeftPanel, ThreadRunnable and RightPanel are second most important classes. LeftPanel holds scenario and RightPanel holds all commands.

They both work in conjunction to provide main functionality to Authorizing App. As seen in the UML diagram below the right panel holds all buttons and their result is shown in left panel. For example when btnStart is pressed ThreadRunnable start() method is invoked and when stop() method is clicked stopRecording() method is invoked which stops recording and offers user to save wav file. Similarly clicking btnMoveUp and btnMoveDown in RightPanel will invoke moveUp() and moveDown() which changes positions of item in LeftPanel.







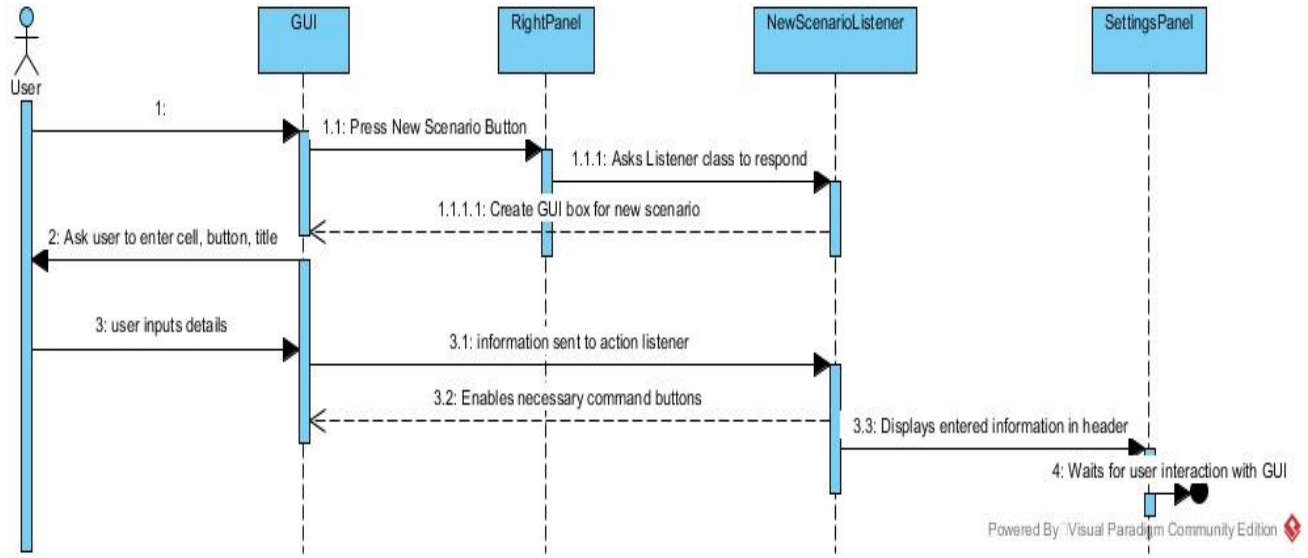
Remaining functionality of RightPanel buttons depends on classes found in listeners package as seen in class diagram. Buttons btnNewScenario, btnImport, btnExport, btnNew, btnNewQuestion corresponds to NewScenarioListener.java, ImportListener.java, ExportListener.java, newButtonListener.java, NewQuestionListener.java classes respectively. These listener classes hold composition relationship with RightPanel as their life cycle depends on life cycle of latter class.

Package commands have composition relationship with NewButtonListener.java class. When btnNew is clicked NewButtonListener.java is invoked which shows all command classes in dialog box as new items that the user can choose and use.

3. Sequence Diagram

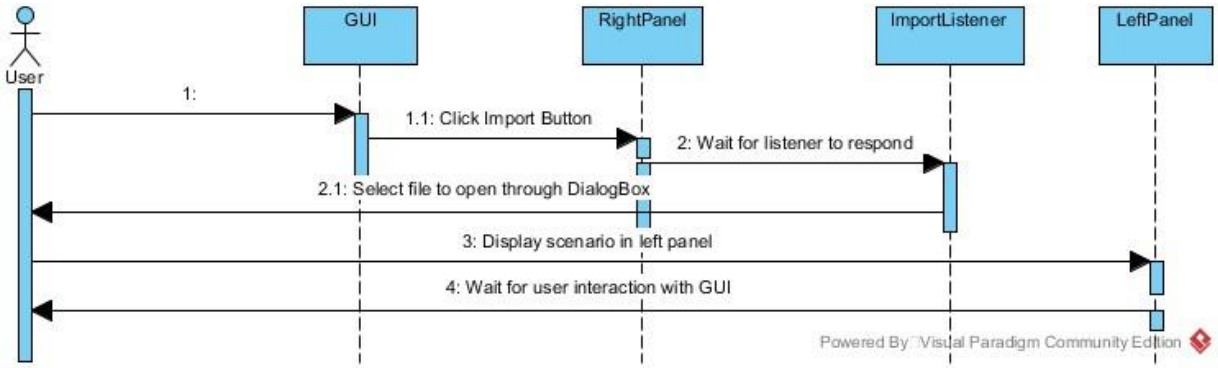
Authorizing Application consists of quiet a few features. Most of these features are associated with RightPanel commands. Some of the features share similar interactions with objects and hence aren't being shown.

First sequence diagram is based on New Scenario Button which as the name suggests creates new scenario and initializes the application.



The user starts by running the application and then clicks the New Scenario Button which invokes RightPanel constructor which in turn waits for NewScenarioListener to respond. NewScenarioListener has scenarioBuilder() method which asks the user to input cell, button and title so that the application can be initialized. After the user inputs information in the interface, the information is sent back to listener scenarioBuilder() method which then initializes other command buttons. The entered information is displayed in header by accessing getSettingsPane() though gui variable.

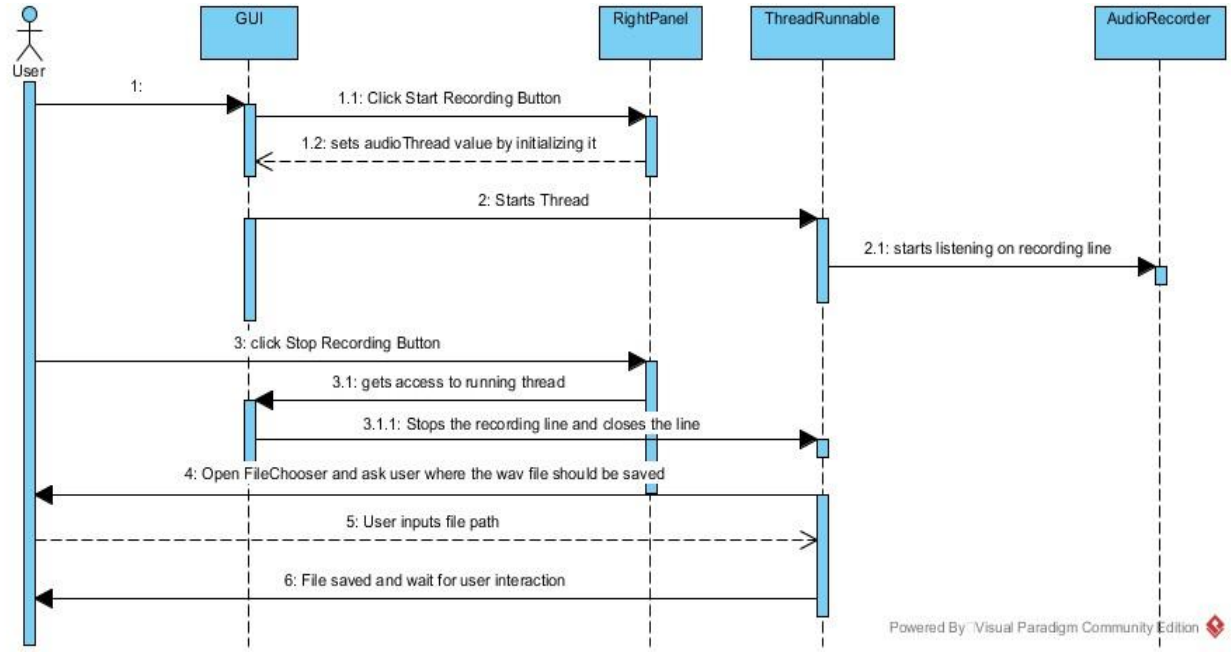
Second sequence diagram is based around import button which shows interaction of GUI, RightPanel, ImportListener, LeftPanel classes.



This is another way to initialize application, as the user can import previous scenarios into the application and make modification on it. When user clicks “Import” button RightPanel constructor is invoked which calls ImportListener. actionPerformed method in listener class opens up a dialog box which asks the user to choose a file that he/she wants to import. When user is done selecting the information is shown in LeftPanel.

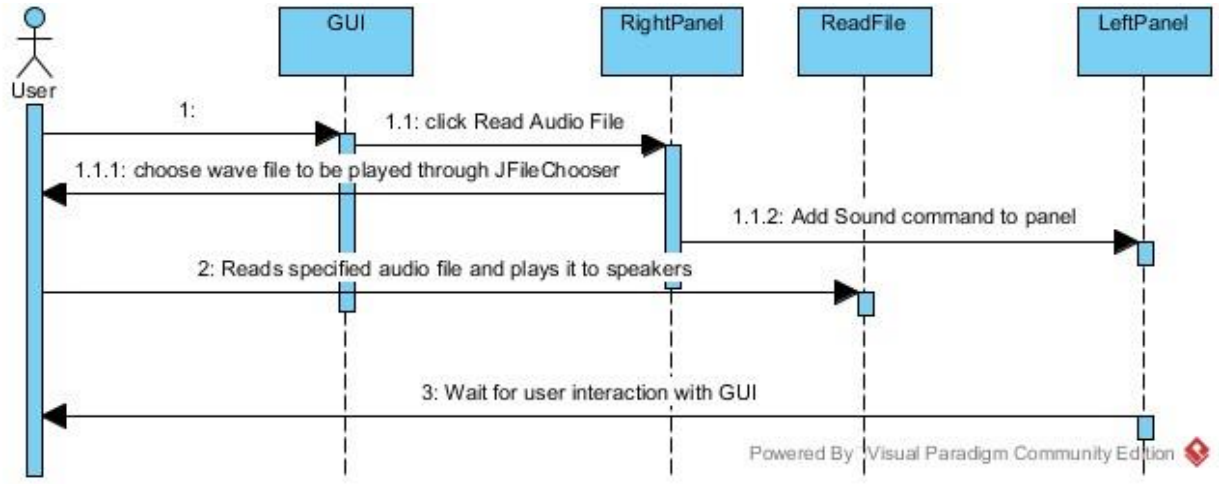
Similar is the case when user clicks Export button. Export button invokes ExportListener and creates an interface that asks user to specify path where the text file must be saved.

Below is sequence diagram showing interaction of classes when user decides to record voice by utilizing “Start Recording” and “Stop Recording” buttons.



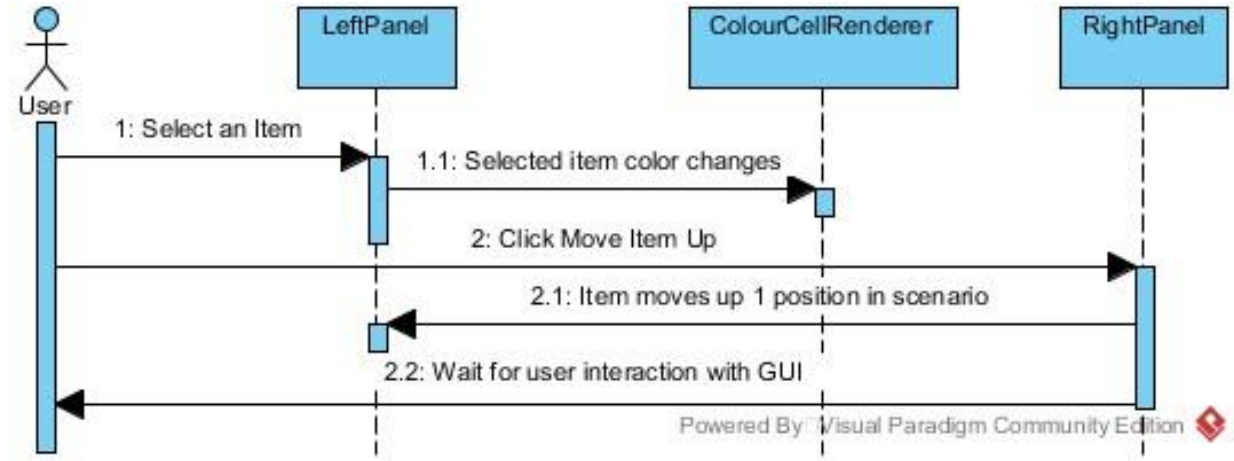
Assuming the application has been initialized, the user clicks “Start Recording” button which sets off actionPerformed method in RightPanel class. This method sets audioThread value by using setAudioThread in GUI class and starts thread with getAudioThread method. Starting thread starts run method in ThreadRunnable which starts recording line using AudioRecorder class. When user clicks “Stop Recording” button actionPerformed is invoked again which goes to ThreadRunnable stopRecording method. stopRecording method stops recording line and creates a wav file. User is asked to specify the path where he/she wants to save wav file.

Sequence diagram below shows interaction between GUI, RightPanel, ReadFile, and LeftPanel classes when “Read Audio File” button is pressed.



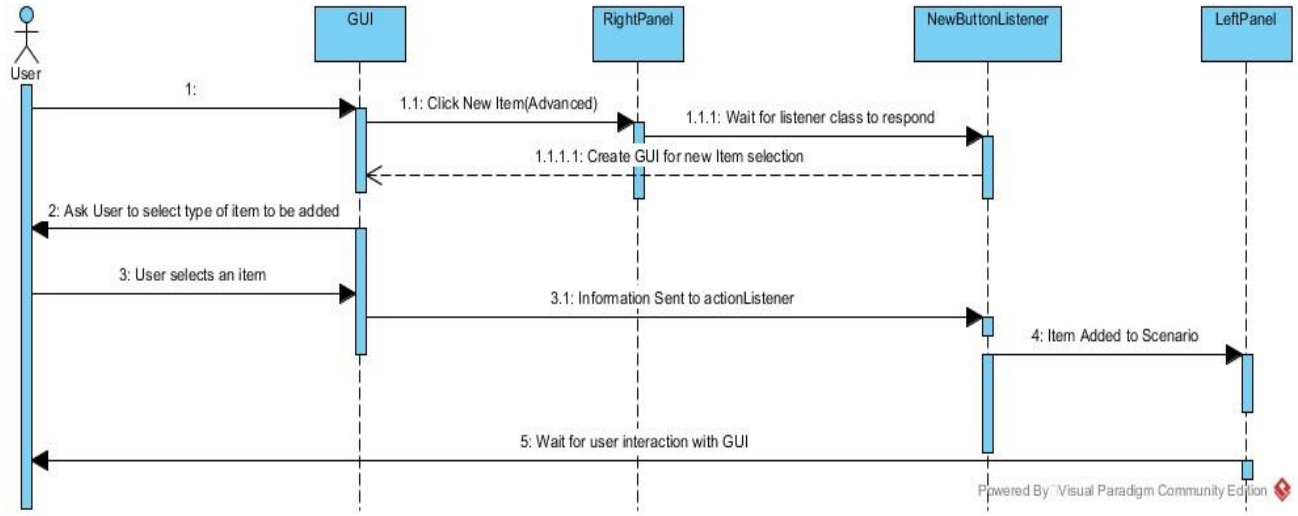
Assuming application has been initialized by creating new scenario or importing scenario, the user clicks “Read Audio File” button. This invoked actionPerformed method in RightPanel which then invoked fileChooser method and the user is asked to specify the wav file he/she wants to be read. This invoked playSound method in ReadFile class and wav file path is added to LeftPanel under Play Sound item.

Sequence diagram below shows LeftPanel, ColorCellRenderer and RightPanel class interact when “Move Item Up” button is pressed.



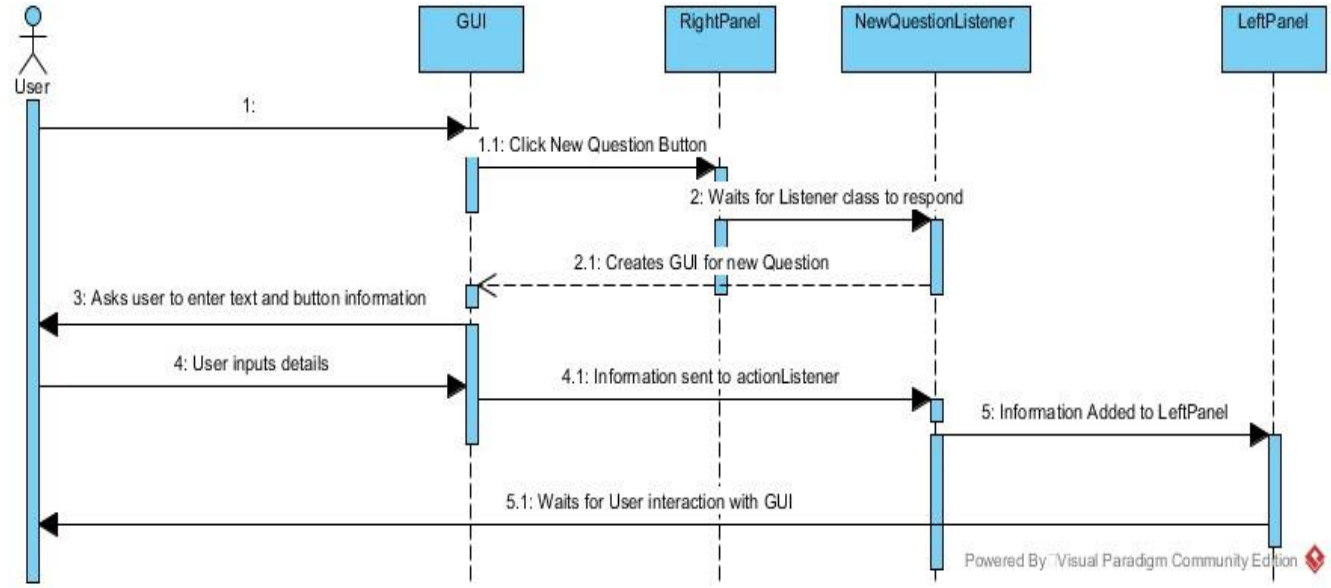
Assuming application has been initialized, the user selects an item in LeftPanel by clicking on it. LeftPanel constructor is invoked which accesses ColorCellRenderer constructor and changes color. User than clicks “Move Item Up” button which invoked actionPerformed method in ightPanel. Then moveUp method in LeftPanel is called the move item in scenario 1 step up.

Sequence diagram below shows GUI, RightPanel, NewButtonListener and LeftPanel class interact when “New Item(advanced)” button is pressed



Assuming application has been initialized, user clicks “New Item” button. This invoked RightPanel constructor which invoked NewButtonListener. Using GUI class an interface is created using which user can select which item to select and add it to scenario.

Sequence diagram below shows GUI, RightPanel, NewQuestionListener and LeftPanel class interact when “New Question” button is pressed



Assuming application has been initialized, user clicks “New Question” button which invokes RightPanel constructor, through which NewQuestionListener is invoked. This listener class creates an interface which asks user for introduction text, braille text, correct button, and text for

incorrect. After user enters the information, it is sent to LeftPanel and is displayed under scenario.

4. Maintenance Scenario

By designing our authoring application with an aim towards modularity, and having a client-supplier interaction between classes, we feel that our application will not be difficult to maintain. The function of the authoring app, such as all the commands that make up a scenario, audio recording, and error checking are encapsulated in different classes that must interact with each other through an API. Such a design is easier to change and modify than one where the functions are implemented in one or a few classes.

For bug fixes, changes can be made to the affected class, and as long as the API that it uses to interact with other parts of the application is not changed, it won't affect the function of the rest of the application. One example of this would be if the default size of the application window needed to be changed, just fields in the GUI.java class need to be changed, and this won't affect any other classes. Another example would be changing the size of an input field in the 'Add Item' dialog. This would just require changing a JtextField object to a JtextArea object before it is added to the GUI, and again this change won't affect the rest how the rest of the application functions.

For adding new functionality to the application, a new class can be written to perform this function and any interactions it needs to have with existing classes will be done through their API. An example would be adding a new button in the Authoring App commands. This would require adding a button to the LeftPanel class and creating a listener class for this button. As with the bug fixes, this should not affect the other functions of the application.