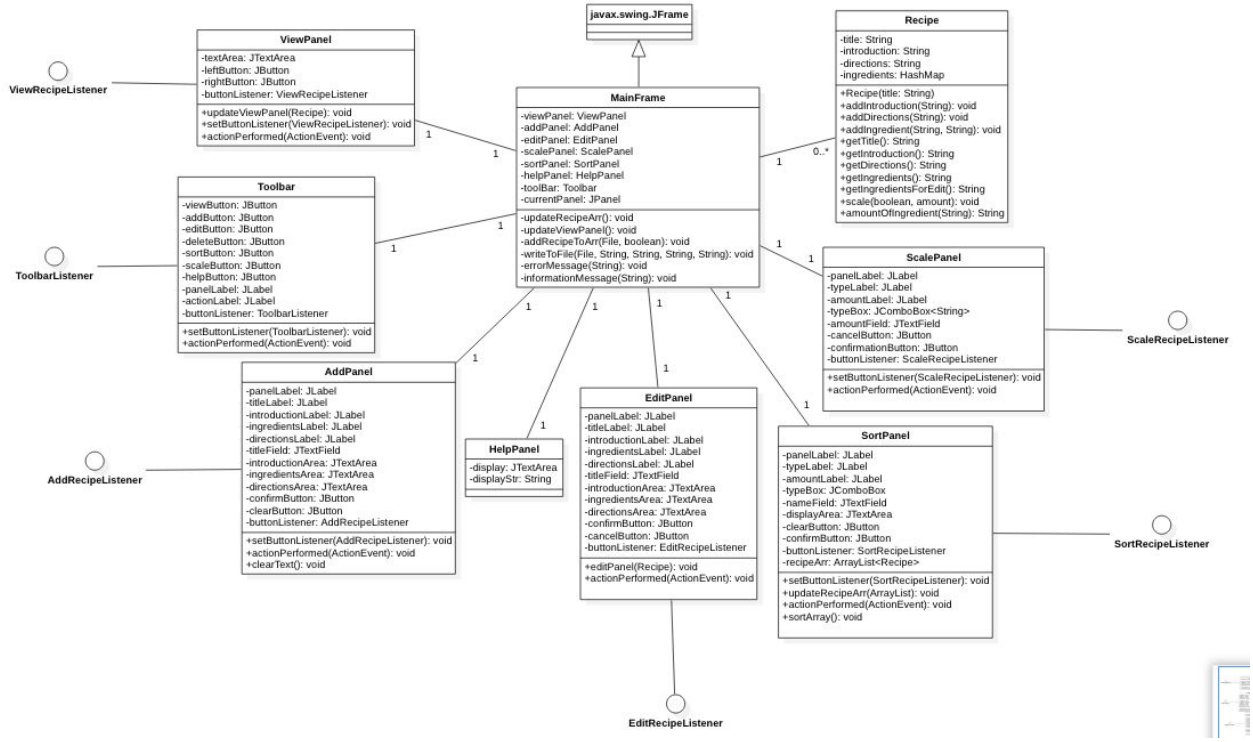
Criterion B: Design

Components of Program

The program will be split into separate modules or panels, and each panel will have its own function. Each recipe will be split into a title, introduction, ingredients, and directions category. All recipes will be written to a text file with its corresponding name being the recipe name and be stored in a **Recipes** folder in the program files. **Recipe Objects** will be made and stored into a **Recipe Array** in the **MainFrame** class for processing recipes.

UML Diagram



*ViewPanel, Toolbar, AddPanel, HelpPanel, EditPanel, SortPanel, and ScalePanel all inherit and extend javax.swing.JPanel, but this was not displayed in the diagram for conciseness. ViewPanel, Toolbar, AddPanel, HelpPanel, EditPanel, SortPanel, and ScalePanel all implement ActionListener, but this was not displayed as for neatness as well.

The structure of the program revolves around having a central **MainFrame** that not handles the central GUI but handles most of the data processing. Other panels are able to interact with the **MainFrame** through the use of public methods of

panel instances or by clever use of interfaces. As shown in the diagram, interfaces, **ViewRecipeListener**, **ToolbarListener**, **AddRecipeListener**, **EditRecipeListener**, **SortRecipeListener**, and **ScaleRecipeListener** are associated with respective "panel classes," but the classes do not implement the interfaces. The panel classes have an instance of the interface, and the **MainFrame** implements the interfaces' method via a "setListener" method public method of the panel classes. This allows panel classes to send information to the **MainFrame** and for the **MainFrame** to do most of the processing. The **MainFrame** can then send information back to panel classes via public methods. This allows for more modularity and less complicated code for a big program.

For example, SortPanel could have the following code:

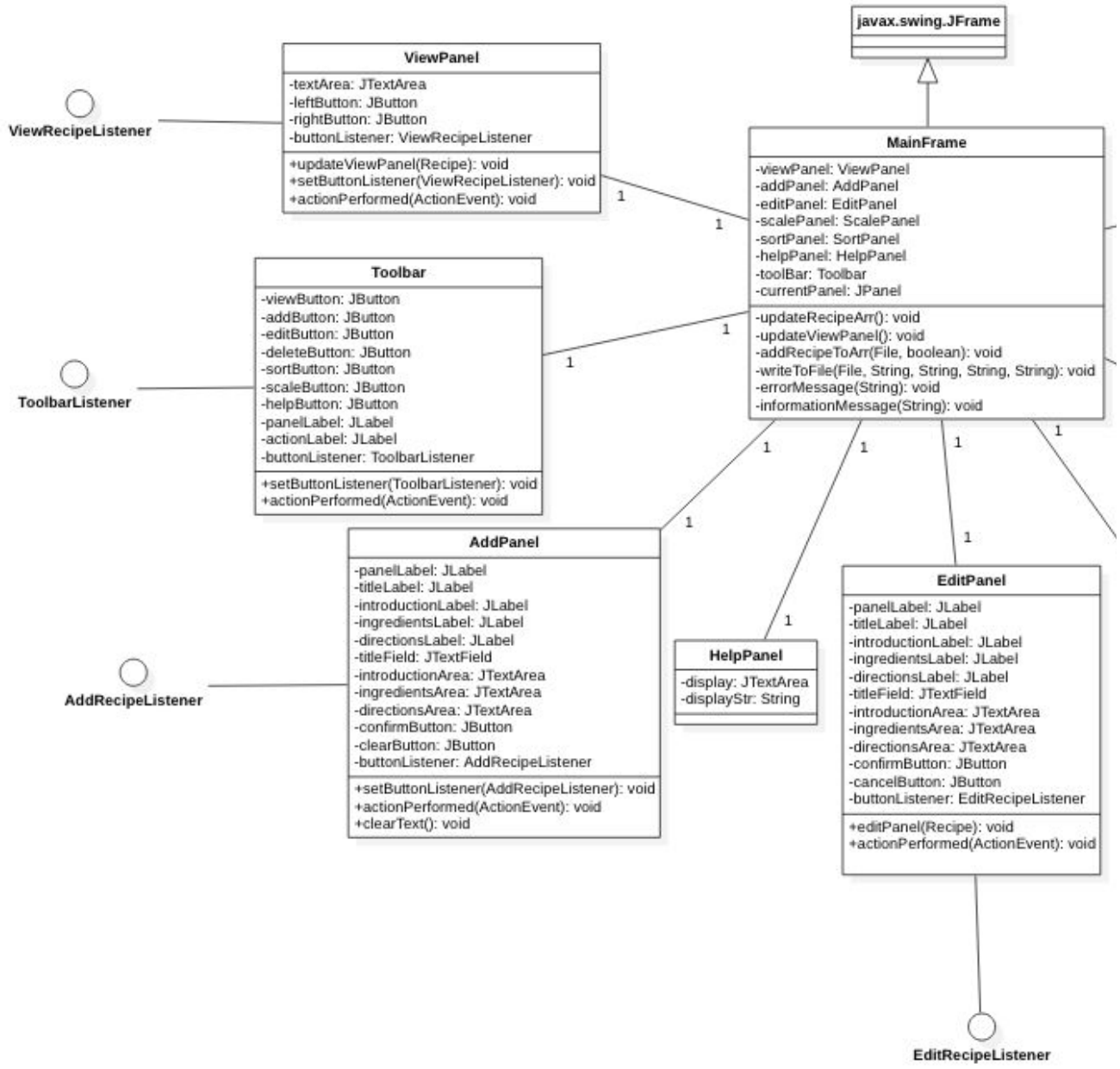
```
private SortRecipeListener buttonListener;

public void setButtonListener(SortRecipeListener buttonListener) { this.buttonListener =
buttonListener; }
```

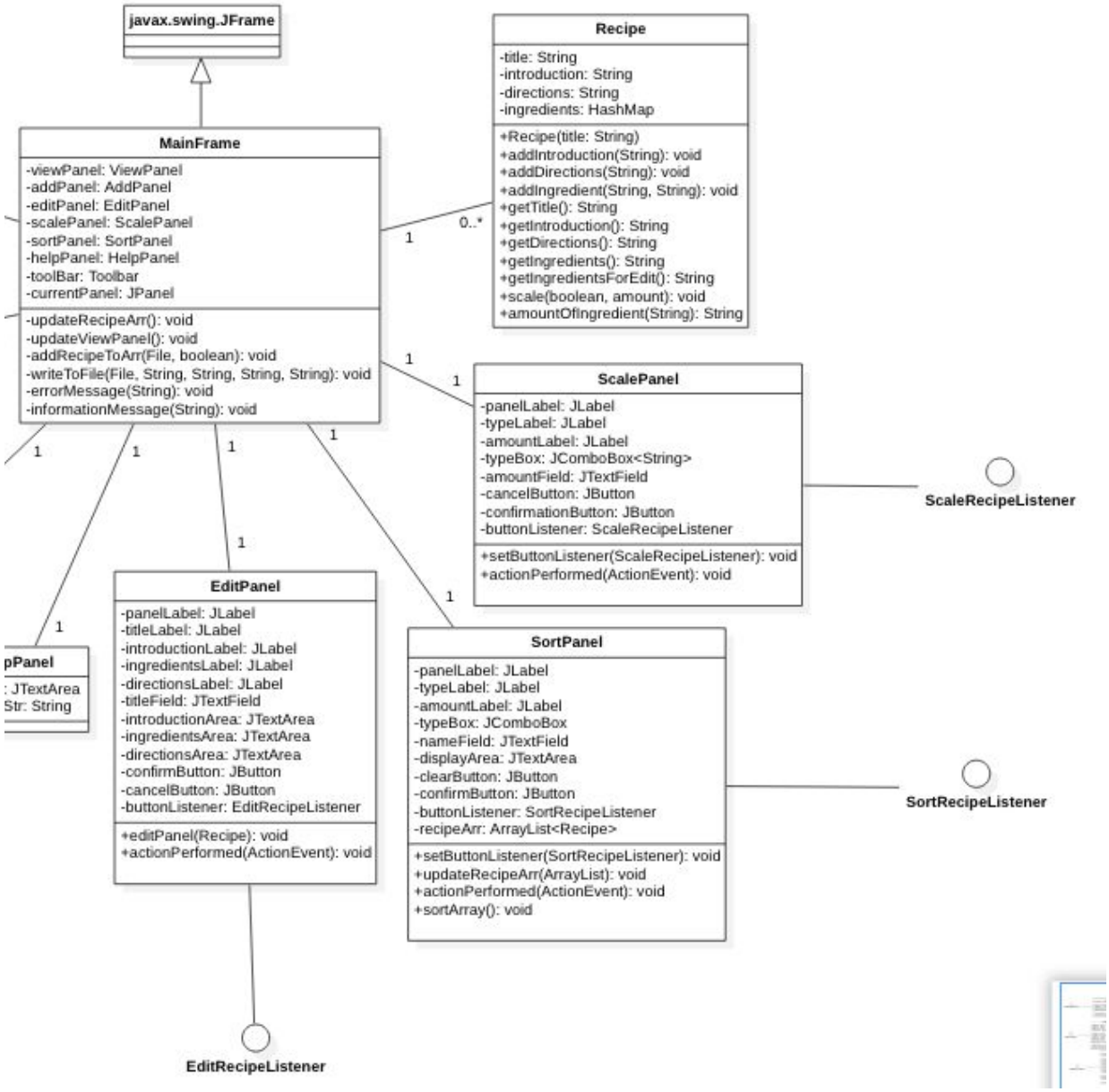
MainFrame implements the required interface method of sortRecipe():

```
sortPanel.setButtonListener(new SortRecipeListener() {
            public void sortRecipe() {
                //Whatever we want to do
            }
        });
```

SortPanel can then call the **sortRecipe()** method implemented in **MainFrame** by doing **buttonListener.sortRecipe()**.

Enlarged UML Diagram

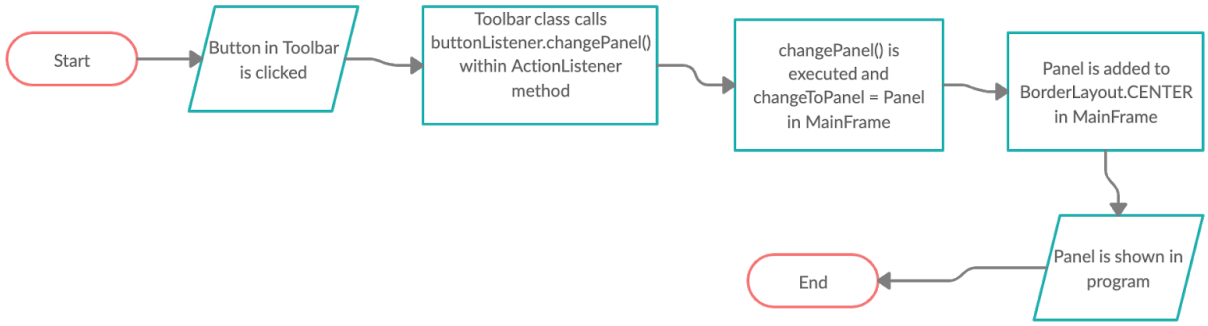



javax.swing.JFrame
Recipe
-title: String
-introduction: String
-directions: String
-ingredients: HashMap
+Recipe(title: String)
+addIntroduction(String): void
+addDirections(String): void
+addIngredient(String, String): void
+getTitle(): String
+getIntroduction(): String
+getDirections(): String
+getIngredients(): String
+getIngredientsForEdit(): String
+scale(boolean, amount): void
+amountOfIngredient(String): String
MainFrame
-viewPanel: ViewPanel
-addPanel: AddPanel
-editPanel: EditPanel
-scalePanel: ScalePanel
-sortPanel: SortPanel
-helpPanel: HelpPanel
-toolBar: Toolbar
-currentPanel: JPanel
-updateRecipeArr(): void
-updateViewPanel(): void
-addRecipeToArr(File, boolean): void
-writeToFile(File, String, String, String, String): void
-errorMessage(String): void
-informationMessage(String): void
0..*
1
ScalePanel
-panelLabel: JLabel
-typeLabel: JLabel
-amountLabel: JLabel
-typeBox: JComboBox<String>
-amountField: JTextField
-cancelButton: JButton
-confirmationButton: JButton
-buttonListener: ScaleRecipeListener
+setButtonListener(ScaleRecipeListener): void
+actionPerformed(ActionEvent): void
ScaleRecipeListener
EditPanel
-panelLabel: JLabel
-titleLabel: JLabel
-introductionLabel: JLabel
-ingredientsLabel: JLabel
-directionsLabel: JLabel
-titleField: JTextField
-introductionArea: JTextArea
-ingredientsArea: JTextArea
-directionsArea: JTextArea
-confirmButton: JButton
-cancelButton: JButton
-buttonListener: EditRecipeListener
+editPanel(Recipe): void
+actionPerformed(ActionEvent): void
SortPanel
-panelLabel: JLabel
-typeLabel: JLabel
-amountLabel: JLabel
-typeBox: JComboBox
-nameField: JTextField
-displayArea: JTextArea
-clearButton: JButton
-confirmButton: JButton
-buttonListener: SortRecipeListener
-recipeArr: ArrayList<Recipe>
+setButtonListener(SortRecipeListener): void
+updateRecipeArr(ArrayList): void
+actionPerformed(ActionEvent): void
+sortArray(): void
SortRecipeListener
EditRecipeListener
pPanel
: JTextArea
Str: String

Main Processes Flowcharts

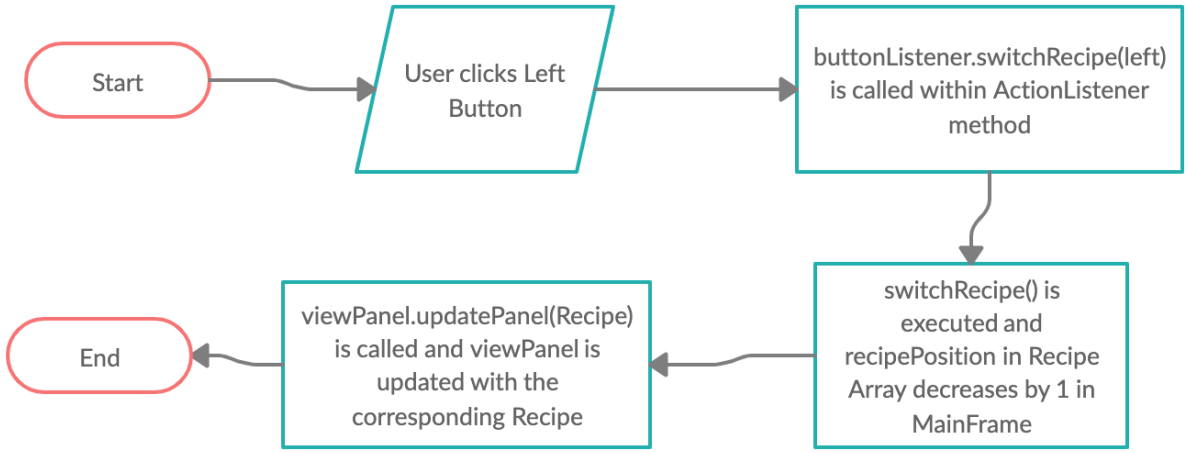
*Some processes have slight variations but are very similar so they are only shown once

Process to Change Panel



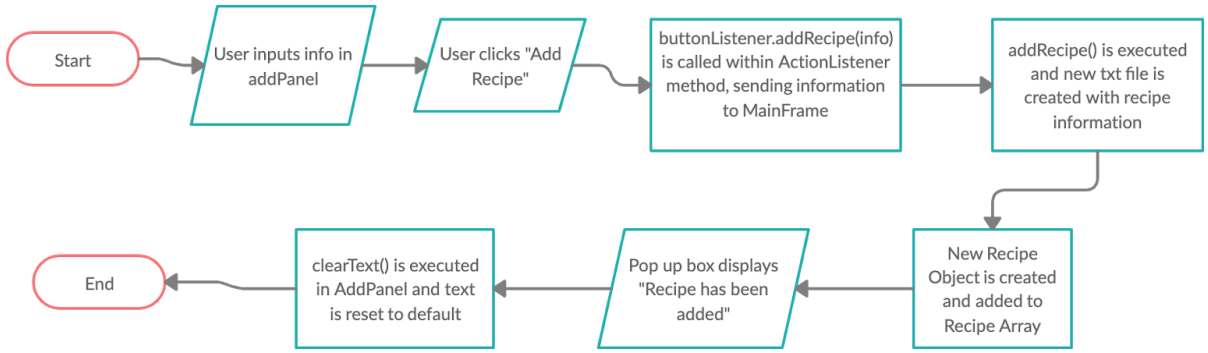
*This process is the same for each panel, but **changePanel(panel)** will change depending on which button is clicked

Process to Switch/Scroll though Recipes via ViewPanel



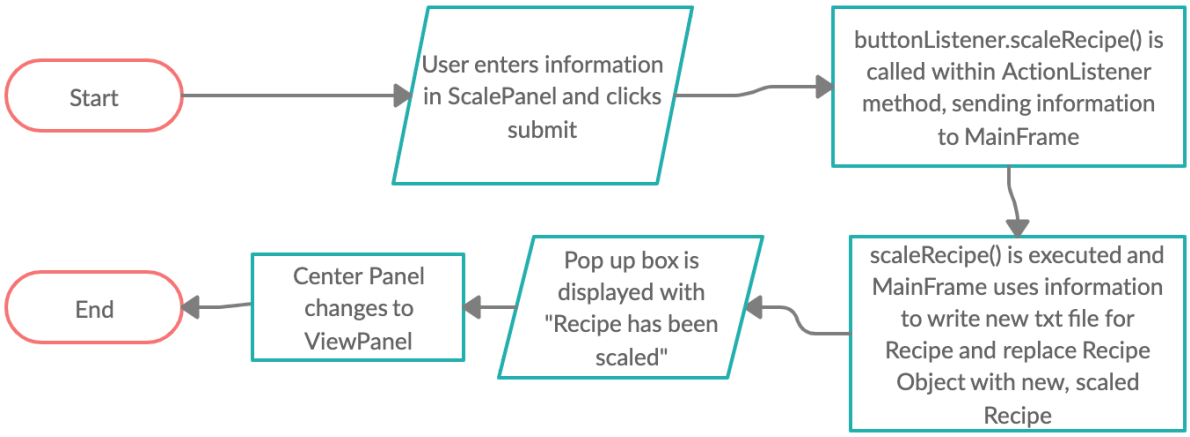
*Process is the same for the Right Button

Process to Add Recipe via AddPanel

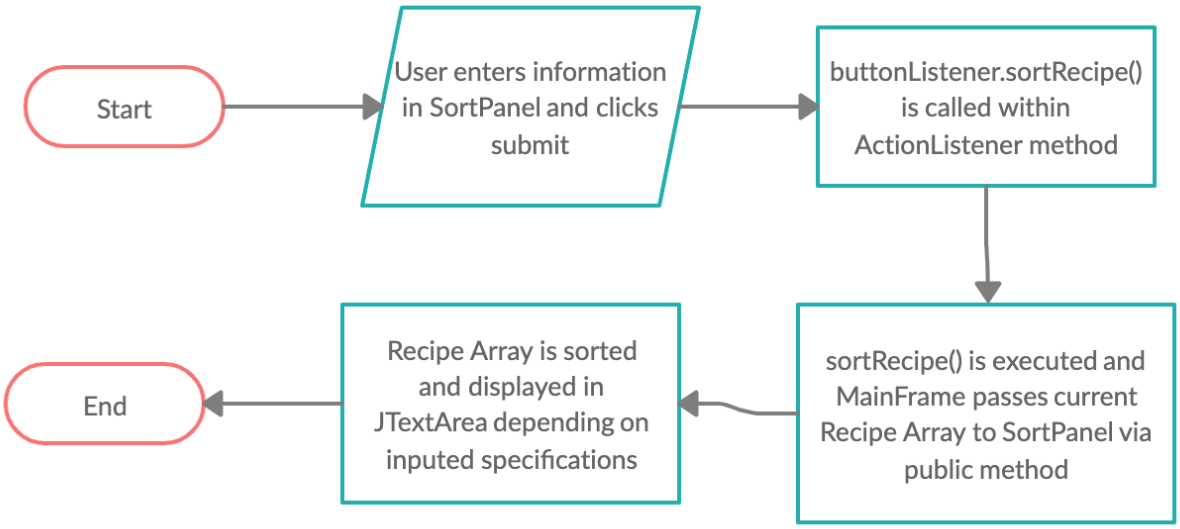


***EditPanel** process to change the recipe is almost exactly the same. The new **Recipe Object** just replaces the existing Recipe instead. The **Delete** process is also the same, but instead of creating a new file, the current file is deleted and the Recipe Object is deleted from the array

Process to Scale Recipes via ScalePanel



Process to Sort/Search Recipes via SortPanel



Sample Pseudocode for Sorting/Searching

For searching for a specific Recipe, a Bubble Sort will be used, bringing the Recipe with the same name to the top

```
bubbleSort(list : array of items, n : length of list, searchStr : target Recipe name)

for i = 0 to n-1 do:
    swapped = false
for j = 0 to n-i-1 do:
    //Compare Recipe names with compareTo to find the title closest to target name
    if Math.abs(searchStr.compareTo(list[j]) > Math.abs(searchStr.compareTo(list[j+1]) then
        swap( list[j], list[j+1] )
        swapped = true
    end if
end for

    if(not swapped) then
        break
    end if
end for
```

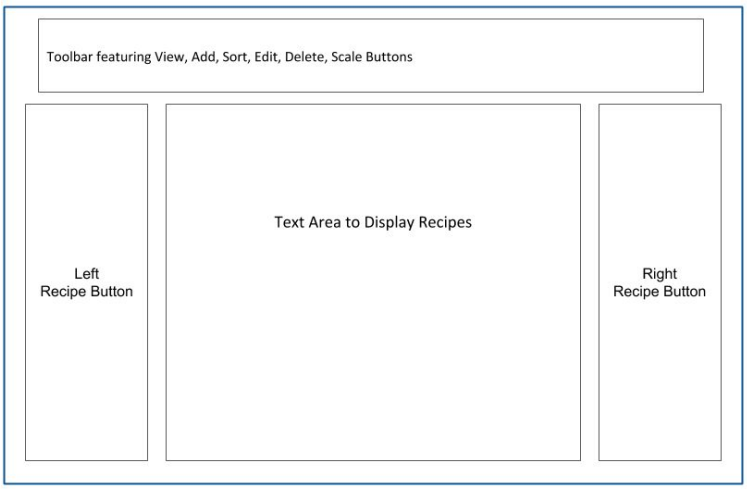
For searching for ingredient, a simple linear search will be done

```
for(Recipe r : recipeArray) do:
    //Use Recipe method to see whether recipe has ingredient
    if r.hasIngredient() then
        display recipe and ingredient amount
    else
        append recipe at the end
end for
```

Design of Panels (Graphical User Interface) with Java Swing

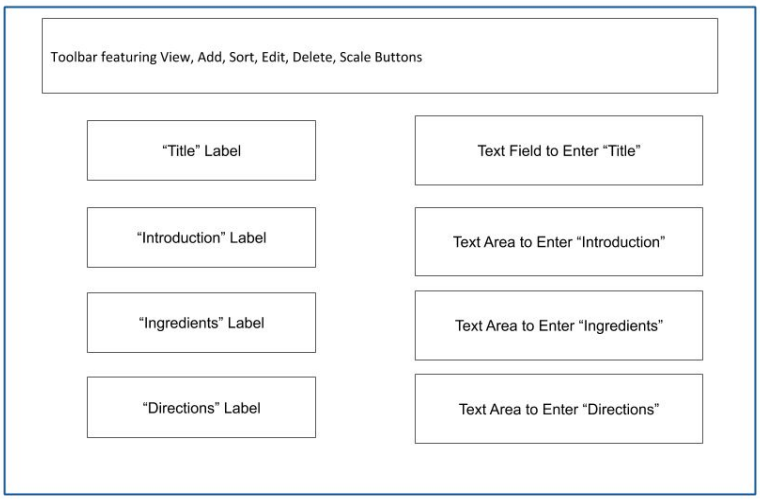
A main, central **JFrame** with **BorderLayout** will house different **JPanels** by swapping **BorderLayout.CENTER** with the appropriate **JPanel**. Each **JPanel** will be made in its own class. Different **JPanels** for each function are shown below. The **"Help"** Panel is simply made up of just a **JTextArea** and is not shown below.
*Submit/Cancel/Clear and subsequent buttons are not shown in the below diagrams but will be added to appropriate panels; Delete and Help buttons will also appear in Toolbar

"View Recipe" Panel



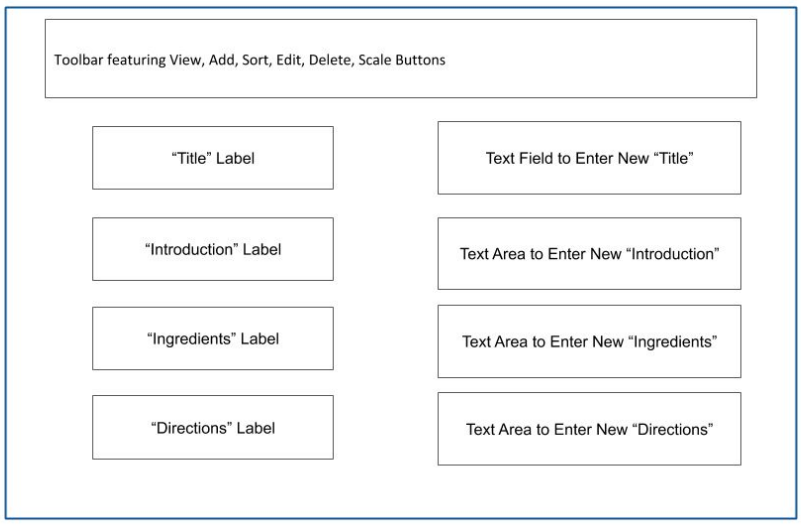
This panel allows the user to view recipes that have been added to the **Recipes** folder and **Recipes Array** of the program. A central **JTextArea** will format and display the recipe and the **Left** and **Right** buttons will toggle through the recipes.

“Add Recipe” Panel



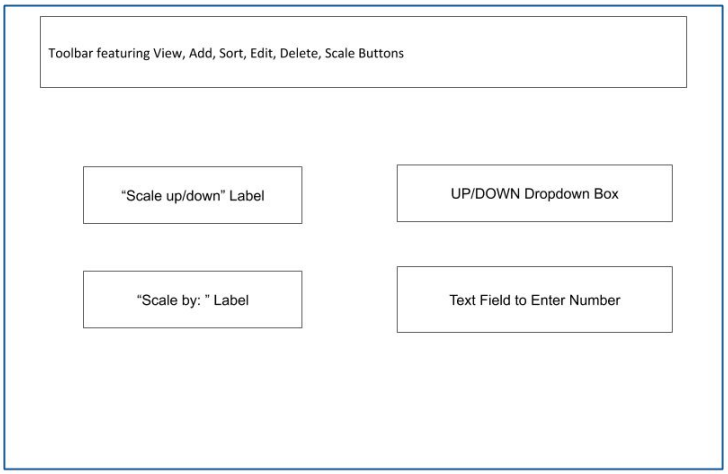
This panel allows the user to input information in order for the recipe to be written to a text file and added to the **Recipes** folder and used throughout the program. There are four categories to put in information regarding the recipe. This panel and similar subsequent panels will use **JLabels**, **JTextFields**, **JTextAreas**, or **JComboBoxes**.

“Edit Recipe” Panel



This panel allows the user to input information in order to change a recipe.

“Scale Recipe” Panel



This panel allows the user to indicate whether they want to scale a recipe’s ingredients up (multiply) or down (divide) and by how much.

“Sort/Search Recipes” Panel

This panel allows users to search up or sort recipes by ingredient or name. User can enter whether they want to sort by ingredient or name and the name of the ingredient or recipe, and the results will be displayed on a **JTextArea**.

Test Plan

Test Type	Nature of Test & Results
Panels should be displayed in correct places and toolbar works	Run program and check panel locations when clicking toolbar buttons
AddPanel correctly adds recipes to a text file and to the program	Input information and click submit; check Recipes folder and Recipes Array to see if recipe is added
EditPanel correctly edits and changes recipes	Input edited recipe information and click submit; check Recipes folder and Recipes Array to see if recipe has changed
Delete button correctly deletes recipe	Click delete and check Recipes folder and Recipe Array to see if recipe has been deleted
ScalePanel correctly scales recipe's ingredients	Input information and click submit; check Recipes folder and Recipes Array to see if recipe has been scaled
SortPanel correctly displays sorted/searched for recipes	Input information and click submit; check if displayed results is logically correct
Pop up messages appear correctly	Put in wrong/abnormal information and click submit - check if error appears; Click submit in different - check if success messages appear
Recipes are displayed in appropriate sections	Go to ViewPanel and check if recipes are split up into correct sections
Recipes can be backed up	Go to Recipes folder and check if added recipes are correct

Words (Extended Writing): 334