Design Document

Union App
Group 13

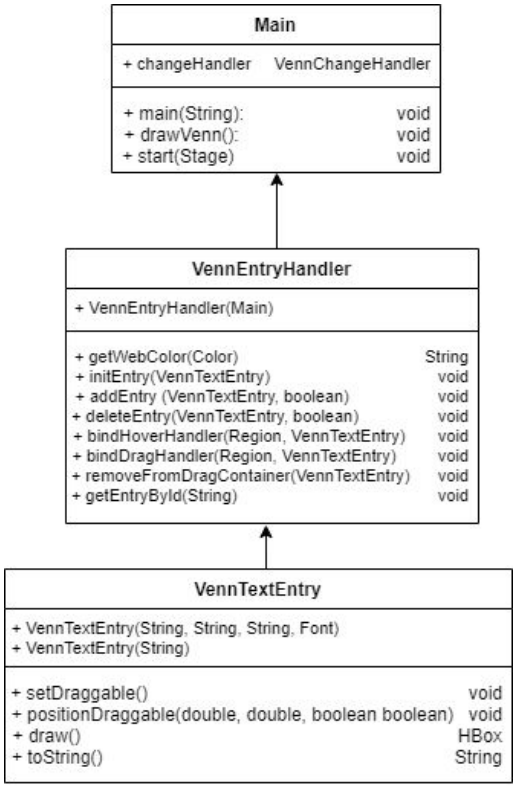
Shivam Patel - 216385247
Colin D'Souza - 216283582
Rushabh Chauhan - 215926108
Sana Khademi - 216326613

Overview:
The Union App allows users to display ideas and data in a visual way, through the use of Venn Diagrams. The most latest version includes new features like adding an entry, dragging/dropping entries, import/export, undo/redo, and other customizable features. Additionally, it fulfills all basic functionality of a digital venn diagram, with a comfortable user interface. Overall, the strategy for completing this project is to break it into several entities that can be represented by Java Classes, each with its own purpose.

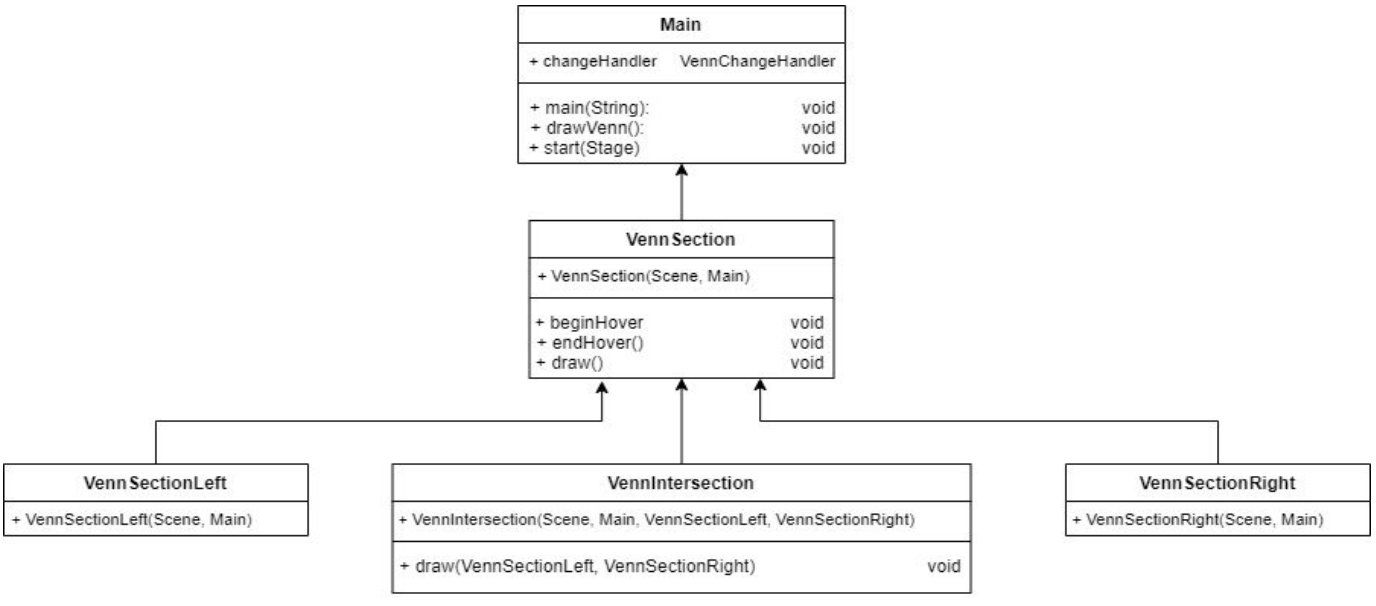
Venn Diagram and Entry Placement:
The most important feature within Union App would be the entries that the user would drag and place on top of the diagram. This feature is broken down into two steps/classes within the project, each responsible for a certain task.

VennEntryHandler is responsible for handling the entries that are added to the customization pane. Furthermore, it keeps track of all entries that are placed on the venn diagram and their locations.

VennTextEntry is responsible for creating the entry based on the specifications (title, description, color, etc..) given by the user. Additionally, it allows the entry made to be draggable by the user.

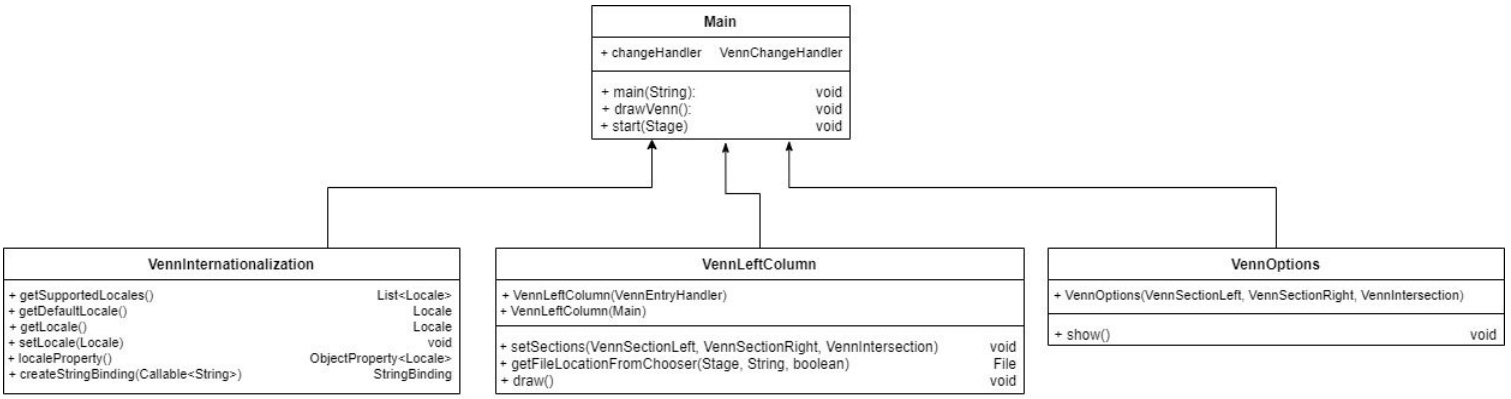


Creating the Venn Diagram Circles:
The most important visual display for this project is the venn diagram and it circles. Once again this component is broken down into three parts (called Sections) which are responsible for displaying that particular section of the venn diagram. *VennSectionLeft, VennIntersection, and VennSection* extend a common class *VennSection*, which altogether creates the VennDiagram seen by the user.

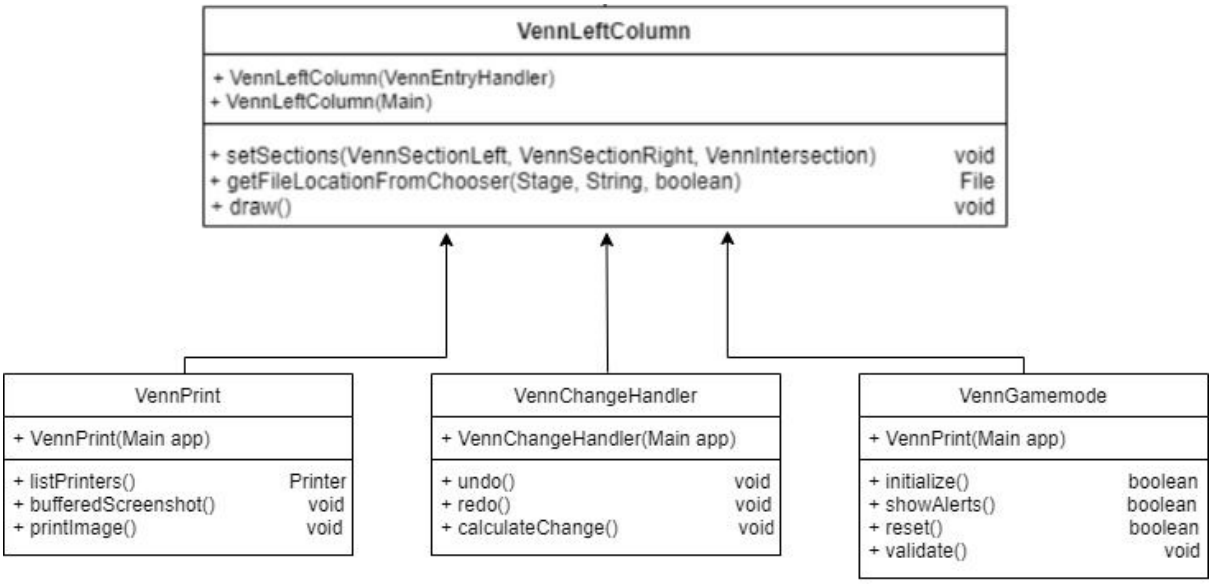


Customization with Union App:

Another important part of the Union App is the customization that is available to the user. In order to display the customization options, there is a customization pane to the left of the diagram. To draw this customization pane, we used a class called *VennLeftColumn*. On the customization pane, there are various buttons available, to perform actions such as adding entries and importing or exporting. The button "Options" presents a menu to the user which allows them to change the colors of the sections of the venn diagram, or change the language of the interface. The options are handled in the class *VennOptions* and the different languages are handled in the class *VennInternationalization*.



Functional Utilities with Union App:

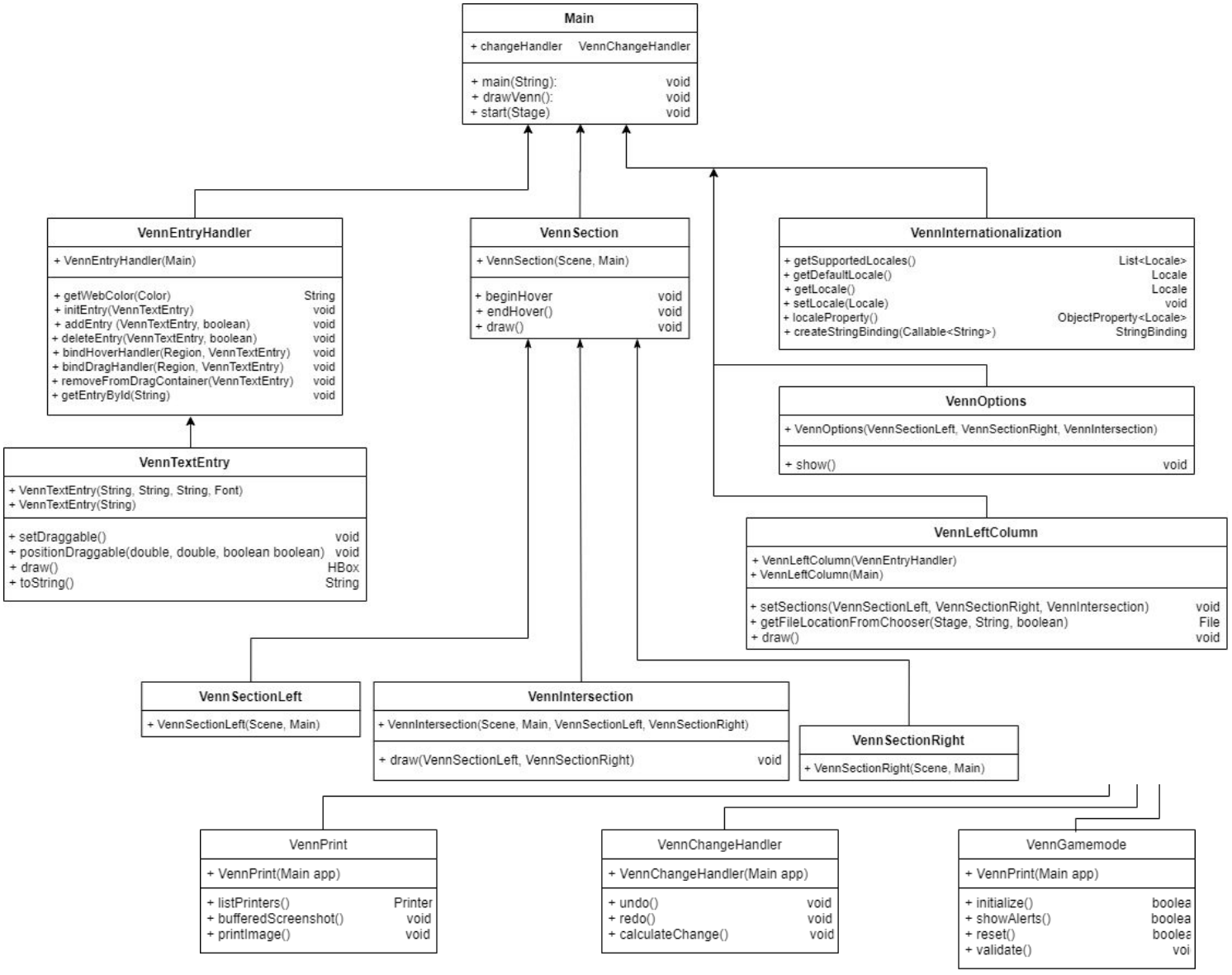


With the updates prioritizing functionality in the application, we added printing and a teacher-friendly gamemode in the classes *VennPrint* and *VennGamemode* respectively. These methods are used and displayed on the customization pane, *VennLeftColumn*.

Undoing and redoing are both also available on the customization pane, and they are handled in the class *VennChangeHandler*. It maintains a history of changes which are used to update the interface accordingly.

Complete Class Diagram:

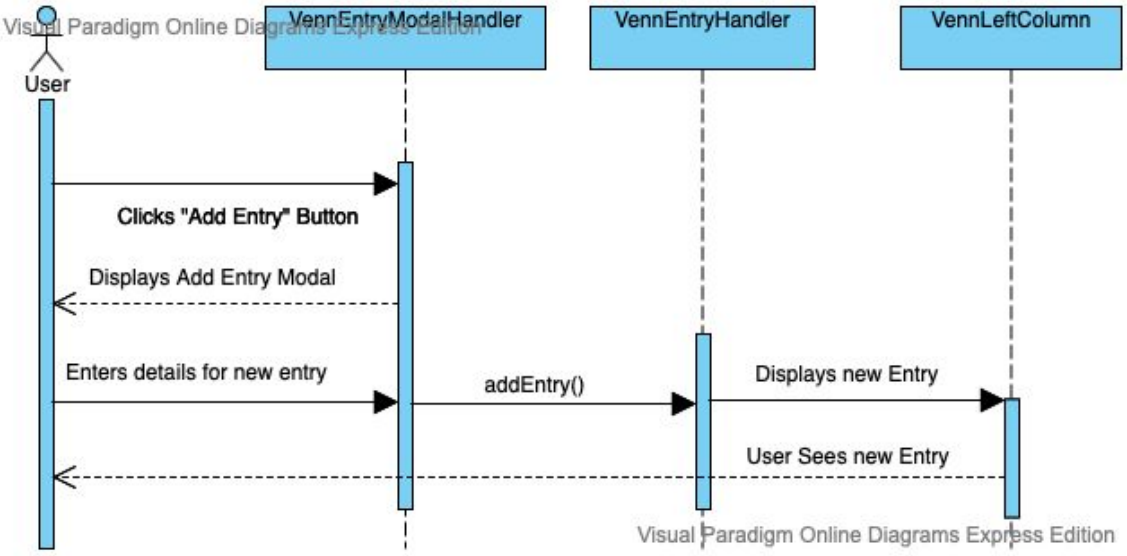
The following diagram shows how all of the above sections combine to power the entire application.



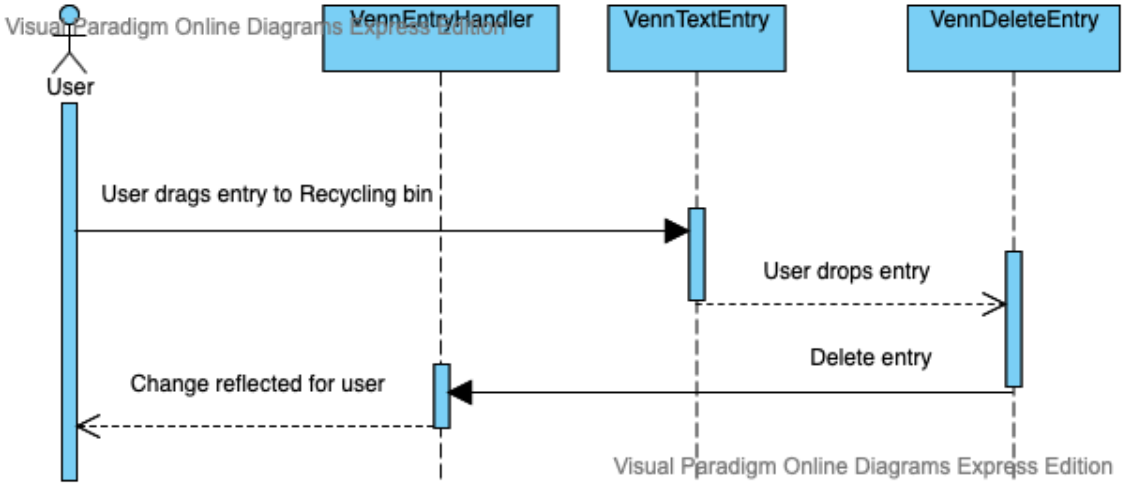
Sequence Diagrams

Here is the description of various scenarios:

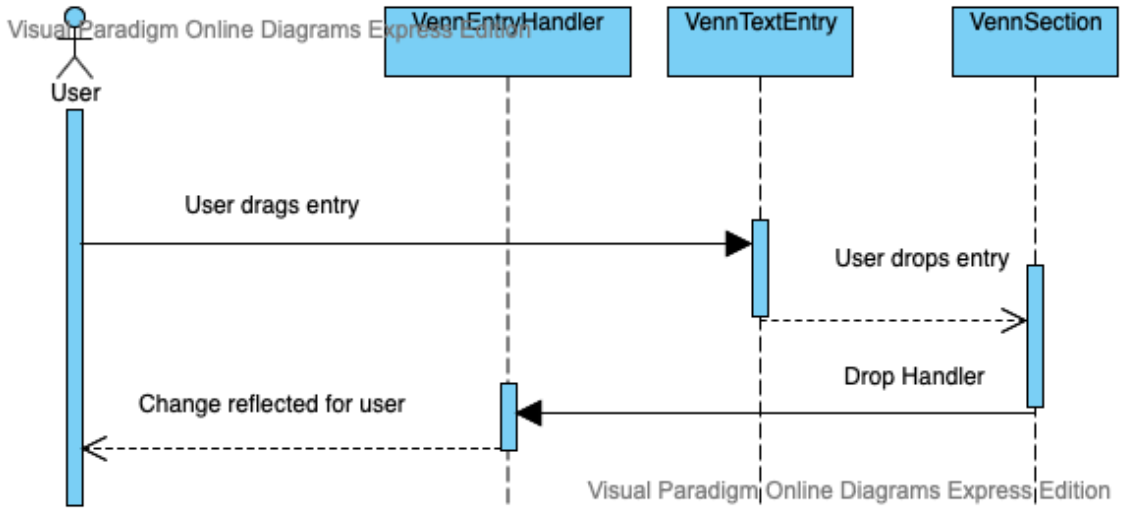
1. Adding entries



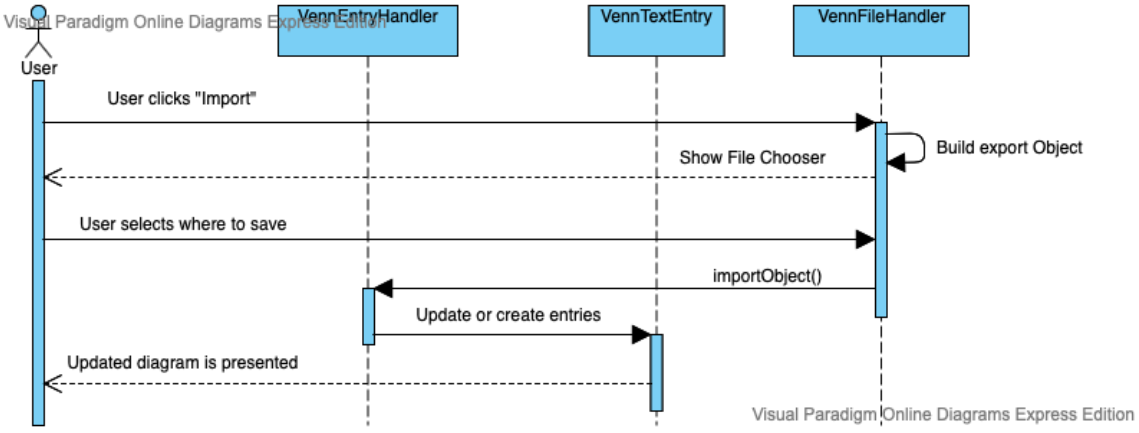
2. Removing entries



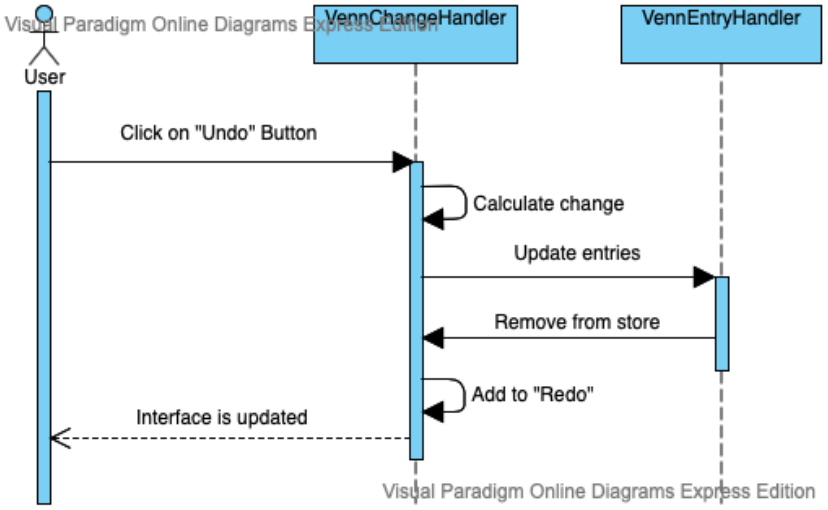
3. Dragging and dropping entries



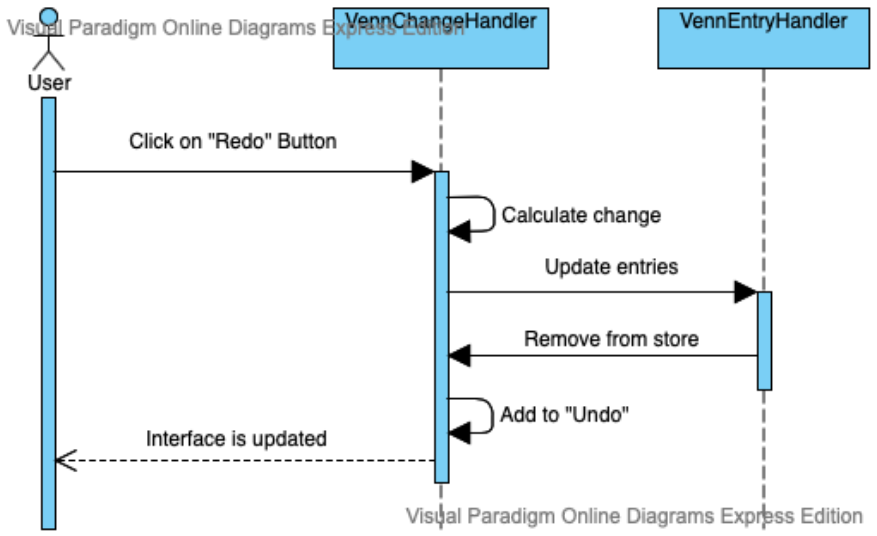
4. Importing venn diagrams



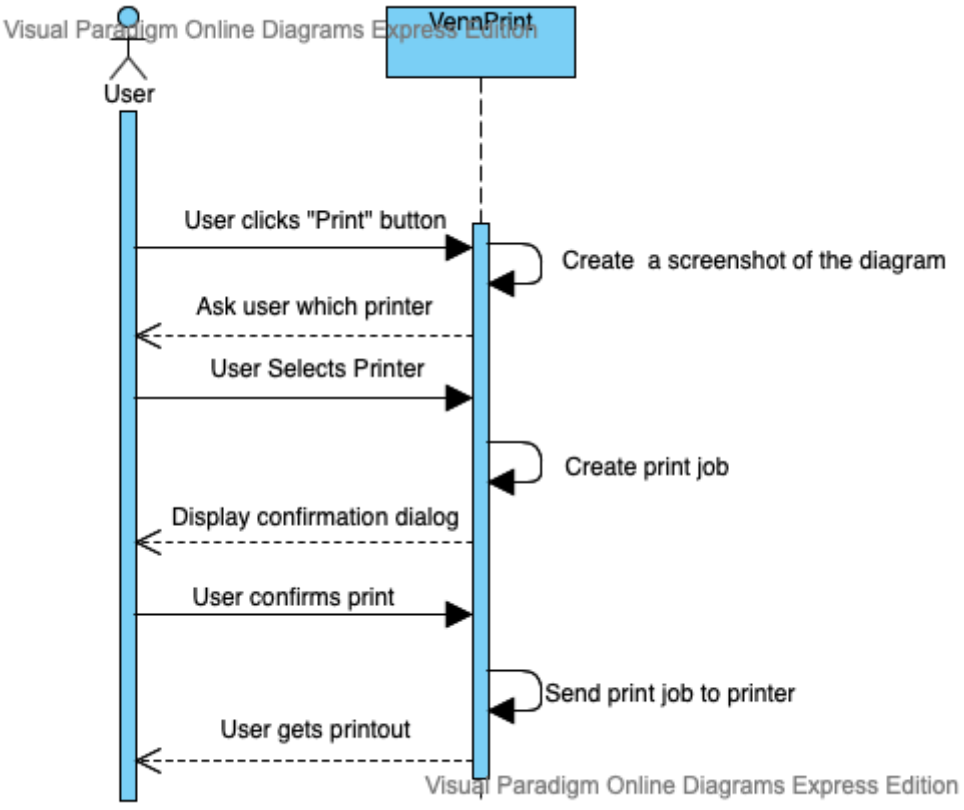
5. Undoing an action



6. Redoing an action



7. Printing



For adding entries, the program relies on the user to enter the details of which they want to add to the diagram (text and zero or more of: color, font, or font size). The program then adds the entry to the central store (available in *VennEntryHandler*) where all the entries are kept, and presented for dragging in the left column (*VennLeftColumn*).

For dragging and dropping entries, the program relies on the JavaFX provided drag and dropping listeners. Upon dragging, the program notes this action and waits for a drop. Upon dropping, the fields of the dragged entry are updated to reflect its new position, and the position in the diagram are updated accordingly.

For deleting entries, the mechanics are very similar to the dragging and dropping the entries, where the user simply drags the entry to the recycling bin on the left to delete.

For importing, the user selects the file they want to import in their file chooser. The program then builds the importing object (available in *VennExport* but omitted in the diagram due to it not having any methods which are called) and updates the store (once again in *VennEntryHandler*) to reflect the changes. The main diagram is updated as well, to reflect the newly imported venn diagram.

Undoing and redoing are changes that rely on a history of the actions performed on the diagram, in the form of a stack. When performing an action, the change is calculated and stored in the stack (in *VennChangeHandler*). When undoing and redoing the changes are pulled and the interface updates accordingly.

Printing relies on a back-and-forth interaction with the user to ensure that the user wants to print their job at the selected printer, and to build and send the print job of the screenshot itself to the printer. The logic for this is in *VennPrint*.

Maintenance Scenarios

Adding Hyperlinks for Venn Titles

This would be a trivial implementation. Implement a checkbox in *VennEntryModalHandler* which would specify if the user wants a checkbox. Then, implement some logic to change the *Label* which is being used under the hood for the entries on the diagram to use the *Hyperlink*. Update the editing modal as well to reflect whether or not there is a hyperlink requested, for future editing.

Adding a third circle

This would be a slightly more intermediate task to complete. The current code relies on the fact for a class for each separate intersection/circle, so for three circles it would need 7 seven different classes in total. The circles are manually positioned, so code for that would be required as well. Simply positioning the circles and fixing the z-index of them should be enough, as the codebase has a centralized class of code for handling the dragging and dropping, which is what each section class extends (*VennSection*). There would also need to be code to add customization for each circle as well (in *VennOptions*). Transitioning from three circles to two would be another hurdle, since the developer would need to ask the user where to put all the elements which are in sections which may be removed.

Adding more Languages

This would be trivial as well. Copy the file named *languages.properties* to a file of the name *languages_iso.properties*, where *iso* in that string is the ISO-639 language code for the language desired. A list is available at https://w.wiki/KzE. Then, edit the file to convert the strings from english to the language desired. Once completed, modify *VennInternationalization* to add the language to the *ArrayList* under the method *getSupportedLocales*. The options menu and the rest of the interface will update automatically.

Resizing of the Circles

The implementation of this is a medium-level task, and would require a good amount of debugging. Updating the static *radius* variable of the *VennSection* class currently updates the size of the circles, though the intersection *VennIntersection* does not resize correctly. There would need to be refactoring to calculate the intersection as the variable changes and to redraw and place the intersection on the interface. There would also need to be calculations done to warn the user that elements inside of the diagram would need to be moved if the desired size to change to is smaller than the location of where the elements are on the Venn diagram.