Cue Sheet App Project

Team members: Patrick Aung, Kevin Ng, Woody Wu

February 26, 2019

I. Project Description

The major goal of this project is to allow users to create, modify and handle bicycle cue sheets by a computer application with a graphical user interface. The interactions between the application and the users are done by buttons and the mouse.

II. Application Features

Following are our application features:

Segment bank display: Shows segments users loaded into the application. Different from canvas display.

Canvas display: Segments can be added/removed to/from the canvas display via the segment bank display. Canvas display shows the current overall route that can be exported as a finalized .csv route file.

Import: Allows users to load a segment in .csv file format into the segment bank display.

Export: Allows the user to export the user-built route from the application's canvas display.

Add: Adds a segment from the segment bank to the canvas display and updates the segment bank.

Remove: Removes a segment from the canvas display and updates the segment bank accordingly.

Drag: Segment vertices can be dragged once segments are added to the canvas display. Clicked vertex is highlighted blue to notify users that current vertex can be dragged.

Save: Provides a mechanism for users to save the current instance of the application by storing the necessary model and GUI info into a .txt file where the user wants to save.

Load: Retrieves the saved .txt file and populates the segment bank and canvas displays.

The specific implementation of each of the features is discussed in the model and GUI classes and interface sections.

III. Model Classes and Interface

The model contains seven classes, which are CSVFileHandler, FileHanderInterface, Listener, Route, Segment, SegmentBank, and SegmentVertex.

CSVFileHandler implements FileHandlerInterface. It helps the system import and export the CSV file.

Considering the flexibility of our project in terms of allowing import and export with multiple file formats, we created FileHandlerInterface. Because of the interface, all we need to do to allow our program to accept a new file format is to add a new class to the FileHandlerInterface.

Segment, an object that represents a segment of the overall route, contains a start and a destination, and an intermediate path.

SegmentVertex represents a start point or a destination of the segment. This is the class that saves the position to be displayed, actual names, and nicknames.

SegmentBank class manages any segments the user imports.

Route class manages all segments the user adds from the segment bank.

Save method relies on the route object used by the application. Since our route object is made up of segmentBankContents, routeContents, and Listener, we first write a private method that turns a segment to its corresponding string. Since listener is fixed, the only information we need to store is those from the segmentBankContents and routeContents. It is important to note that we do not need to worry about our segment bank because segmentBankContents in Route Class updates both its segmentBankContents and the segment bank itself when changes occur. Now, the strategy our team used involves creating three layers of "tags" to differentiate/separate three things: segmentBankContents/routeContents, segment info, segment field info. The three tags for fun are "patrick," "kevin," and "woody." "Patrick works as a separator which separates segmentBankContents and routeContents. "Kevin" marks the start of each segment. "Woody" marks the start of each first of a segment. There will be 2 "patrick"s, any number of "kevin"s based of number of the segments in segmentBankContents and routeContents, and 10 "woody"s for each segment. Following the way the saved.txt file is formed:

```
"patrick
        kevin
                10 woody
patrick
        kevin
```

10 woody”

The load method takes the saved.txt file with patrick,kevin,woody separators and separates them accordingly, turns them back into segments, and adds those segments to segmentBankContents and routeContents. Once segments are added to segmentBankContents and routeContents, the load method sets the contents of the SegmentBank with the contents from segmentBankContents. This process is all that is needed to recreate the application instance that the user saved.

IV. Graphical User Interface Classes and Interface

For the cue sheet project, there are in total of two main displays: the SegmentDisplay and the CanvasDisplay. The SegmentDisplay allows users to import segments from device, label segments and add segments into the overall route. All those functionalities are done by 3 buttons and a SegDisButtonListener. By having several different functioning buttons, an object implemented with button ActionListener interface is necessary, and it is called SegDisButtonListener. On the other hand, the CanvasDisplay shows the most current overall route and allows users to export, save, and load routes with buttons. Given the necessary buttons, the button ActionListener interface is also needed for programming different functioning buttons.

V. Work Distribution

The work distribution of this project is as follows:

Patrick was responsible for Segment, SegmentBank, CSVFileHandler, Save/Load route, and FileHandlerInterface.

Woody was responsible for Segment, SegmentVertex, SegmentBank, Route, and Tests.

Kevin was responsible for all the GUI-related classes.

VI. Class Diagram

The following figure represents the design class diagram for the cue sheet program version 1:

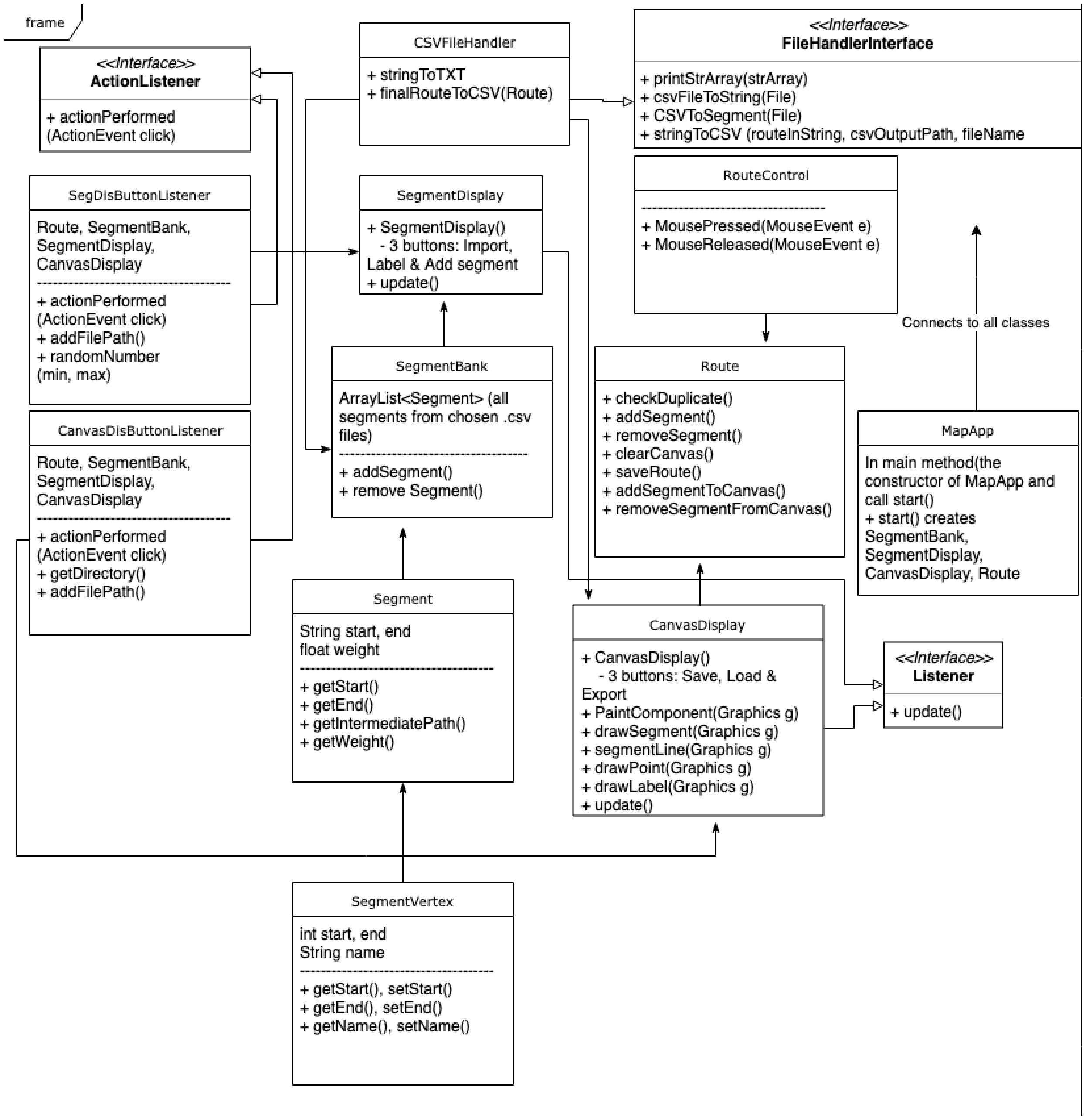

frame
<<Interface>> ActionListener
+ actionPerformed (ActionEvent click)
CSVFileHandler
+ stringToTXT
+ finalRouteToCSV(Route)
<<Interface>> FileHandlerInterface
+ printStrArray(strArray)
+ csvFileToString(File)
+ CSVToSegment(File)
+ stringToCSV (routeInString, csvOutputPath, fileName
SegDisButtonListener
Route, SegmentBank, SegmentDisplay, CanvasDisplay
+ actionPerformed (ActionEvent click)
+ addFilePath()
+ randomNumber (min, max)
SegmentDisplay
+ SegmentDisplay()
- 3 buttons: Import, Label & Add segment
+ update()
RouteControl
+ MousePressed(MouseEvent e)
+ MouseReleased(MouseEvent e)
Connects to all classes
SegmentBank
ArrayList<Segment> (all segments from chosen .csv files)
+ addSegment()
+ remove Segment()
Route
+ checkDuplicate()
+ addSegment()
+ removeSegment()
+ clearCanvas()
+ saveRoute()
+ addSegmentToCanvas()
+ removeSegmentFromCanvas()
CanvasDisButtonListener
Route, SegmentBank, SegmentDisplay, CanvasDisplay
+ actionPerformed (ActionEvent click)
+ getDirectory()
+ addFilePath()
MapApp
In main method(the constructor of MapApp and call start()
+ start() creates SegmentBank, SegmentDisplay, CanvasDisplay, Route
Segment
String start, end
float weight
+ getStart()
+ getEnd()
+ getIntermediatePath()
+ getWeight()
CanvasDisplay
+ CanvasDisplay()
- 3 buttons: Save, Load & Export
+ PaintComponent(Graphics g)
+ drawSegment(Graphics g)
+ segmentLine(Graphics g)
+ drawPoint(Graphics g)
+ drawLabel(Graphics g)
+ update()
<<Interface>> Listener
+ update()
SegmentVertex
int start, end
String name
+ getStart(), setStart()
+ getEnd(), setEnd()
+ getName(), setName()

Figure 1: The design class diagram of cue sheet program V1

VII. Interaction Diagram

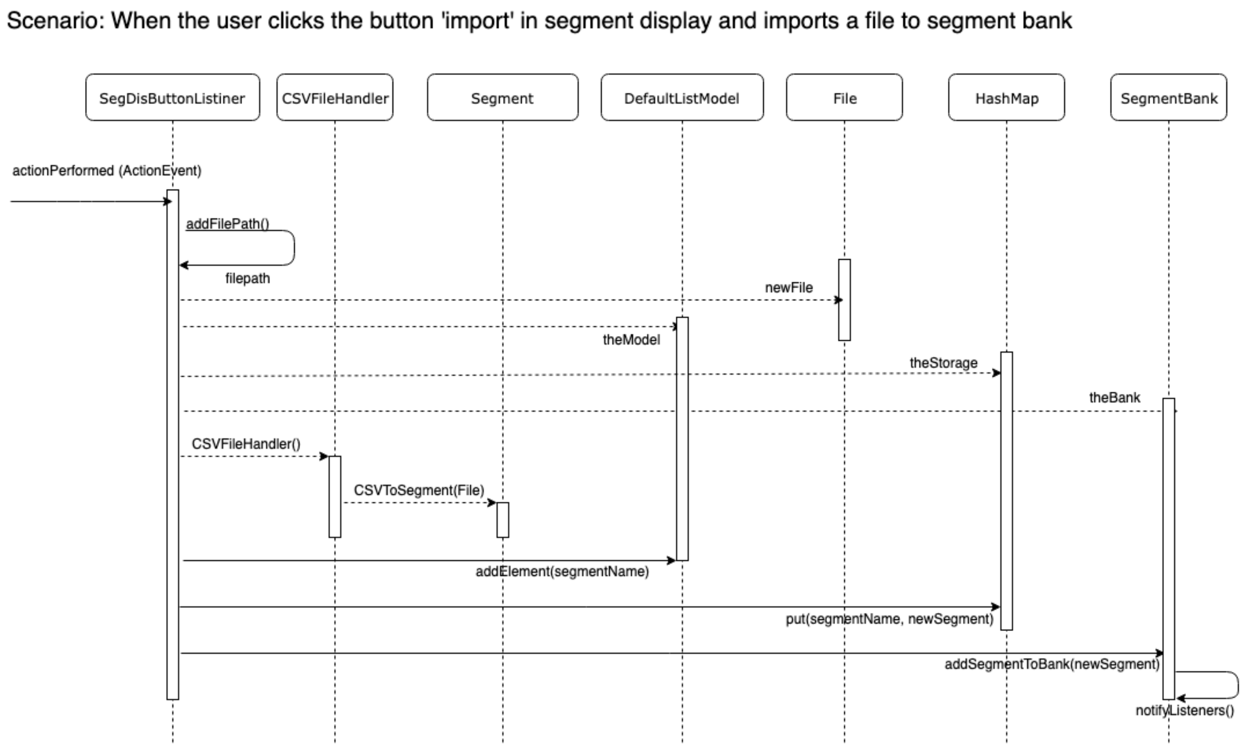


Figure2. The interaction diagram of Import

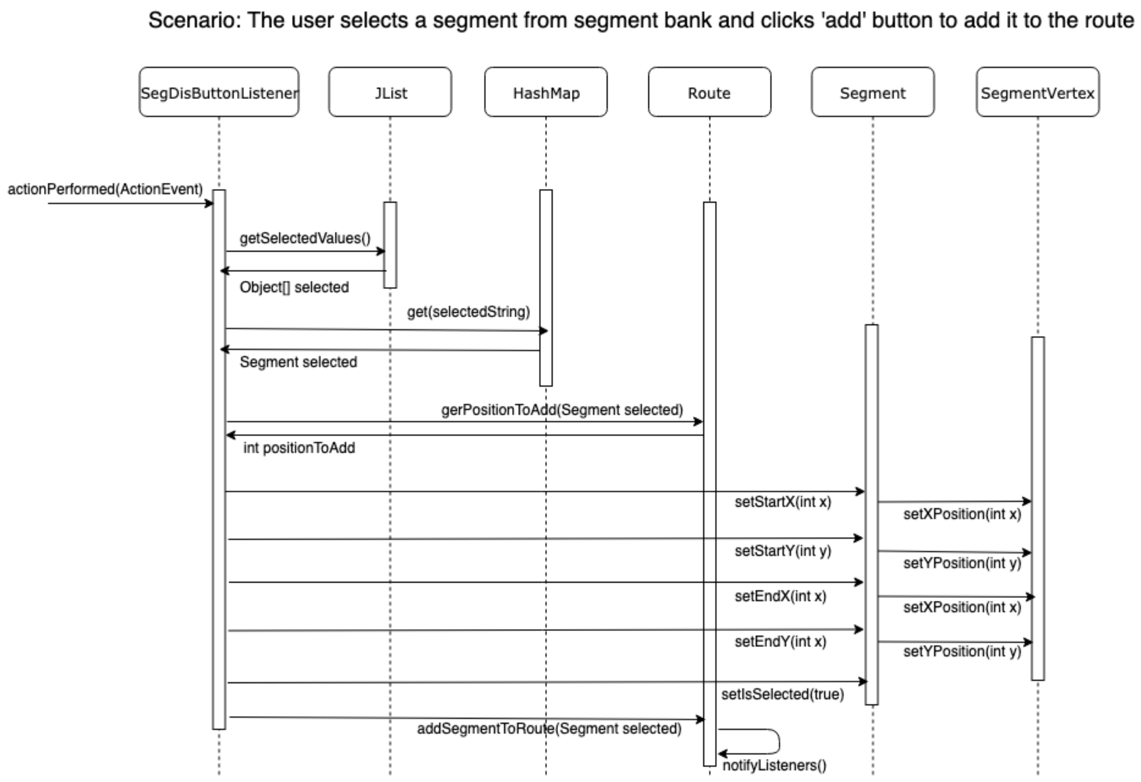


Figure3. The interaction diagram of Add

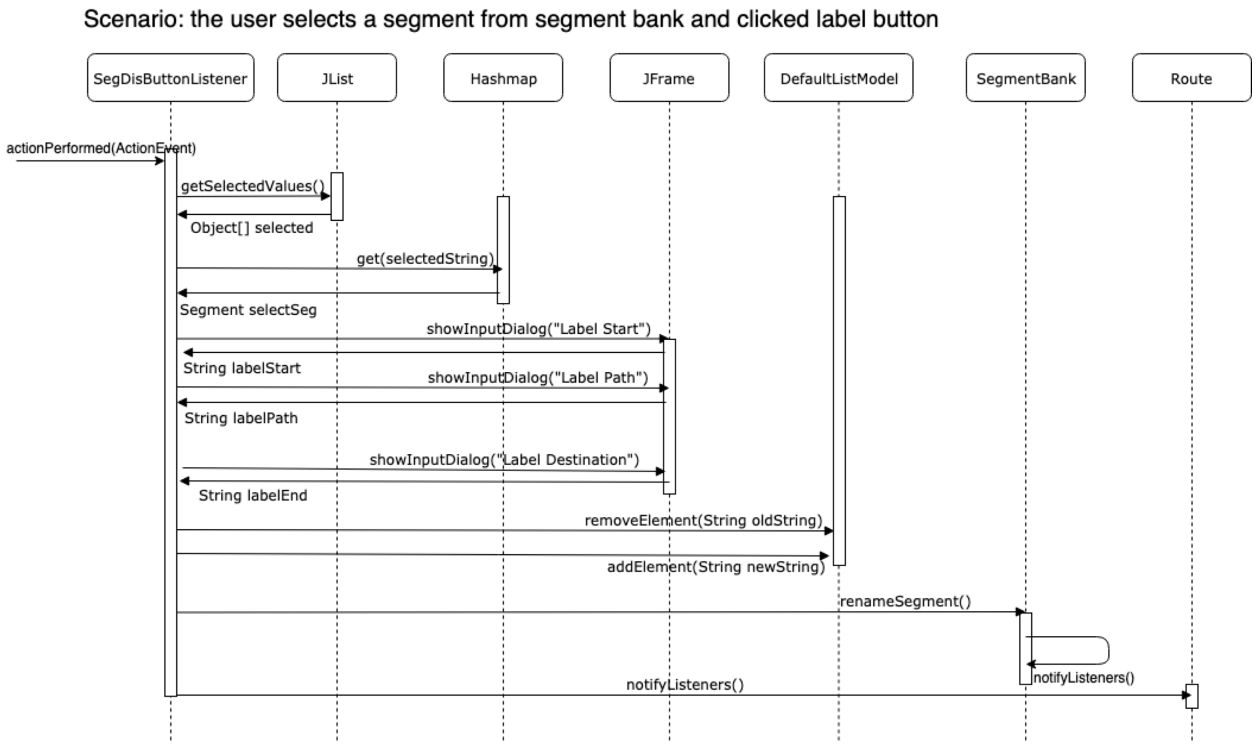

Scenario: the user selects a segment from segment bank and clicked label button
SegDisButtonListener
JList
Hashmap
JFrame
DefaultListModel
SegmentBank
Route
actionPerformed(ActionEvent)
getSelectedValues()
Object[] selected
get(selectedString)
Segment selectSeg
showInputDialog("Label Start")
String labelStart
showInputDialog("Label Path")
String labelPath
showInputDialog("Label Destination")
String labelEnd
removeElement(String oldString)
addElement(String newString)
renameSegment()
notifyListeners()
notifyListeners()

Figure4. The interaction diagram of Label

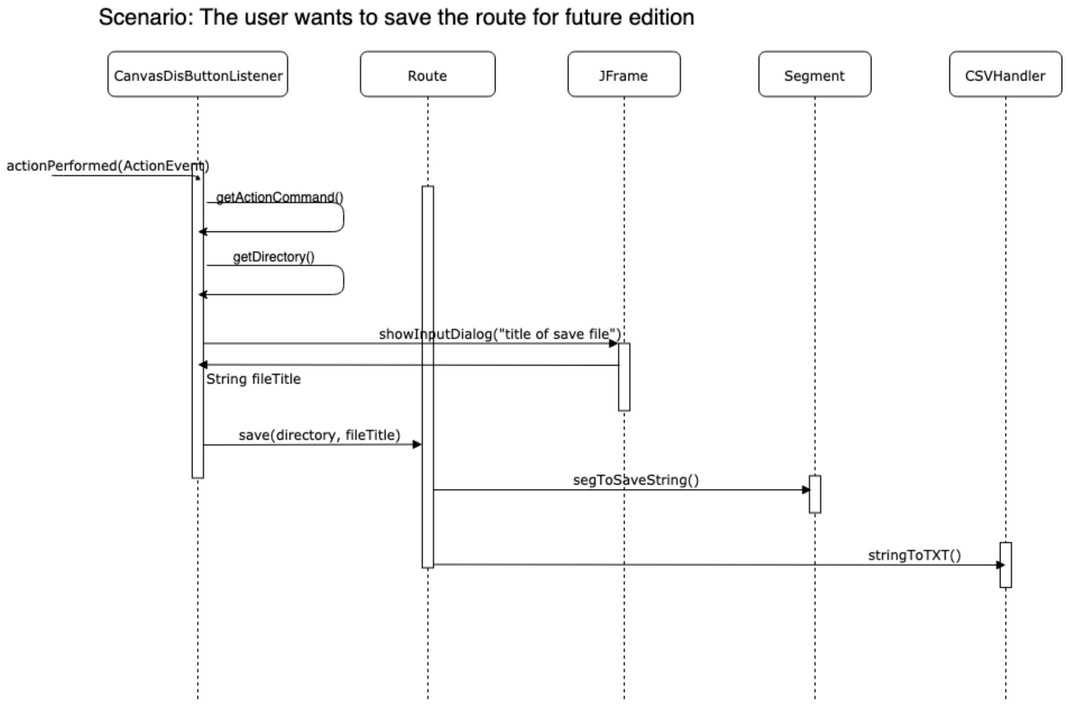

Scenario: The user wants to save the route for future edition
CanvasDisButtonListener
Route
JFrame
Segment
CSVHandler
actionPerformed(ActionEvent)
getActionCommand()
getDirectory()
showInputDialog("title of save file")
String fileTitle
save(directory, fileTitle)
segToSaveString()
stringToTXT()

Figure5. The interaction diagram of Save

Scenario: The user wants to work on the file that was saved previously

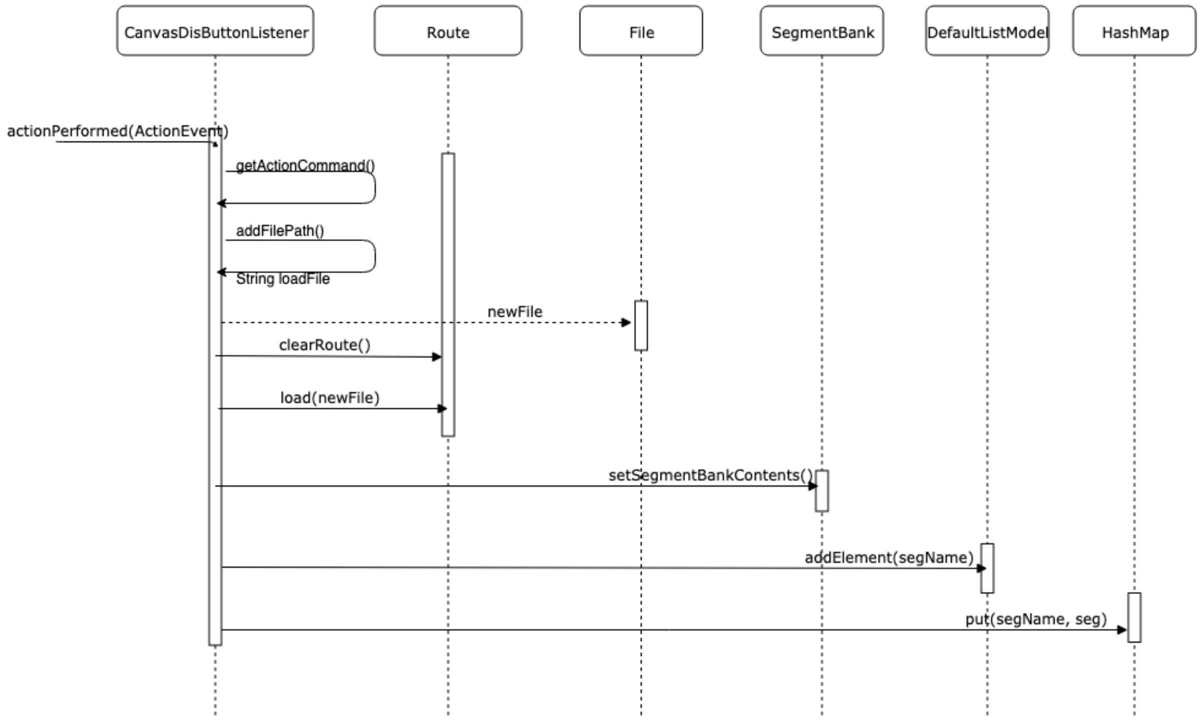


Figure6. The interaction diagram of Load

Scenario: The user finishes the route and exports it as csv file

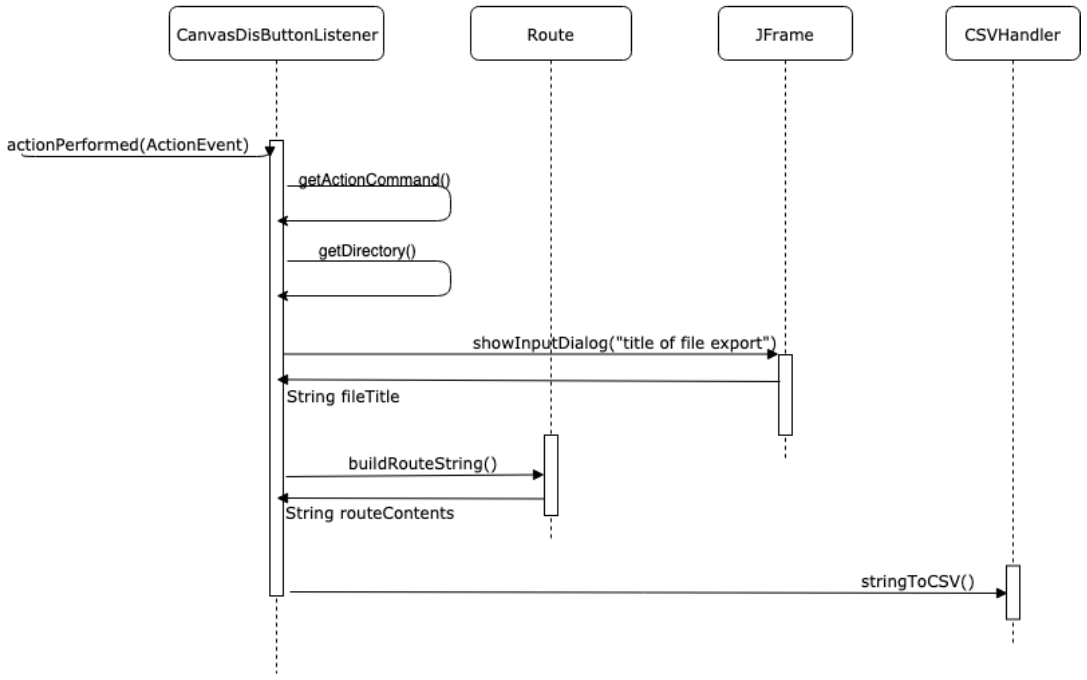


Figure7. The interaction diagram of Export

VIII. Testing

In order to ensure the functionality of classes in the model before merging them with other classes, we wrote RouteTest and SegmentTest to test if our model behaved as expected.

RouteTest:

- testAddSegmentToRoute
- testRemoveSegment
- testGetPositionToAdd
- testClearRoute
- testBuildToString

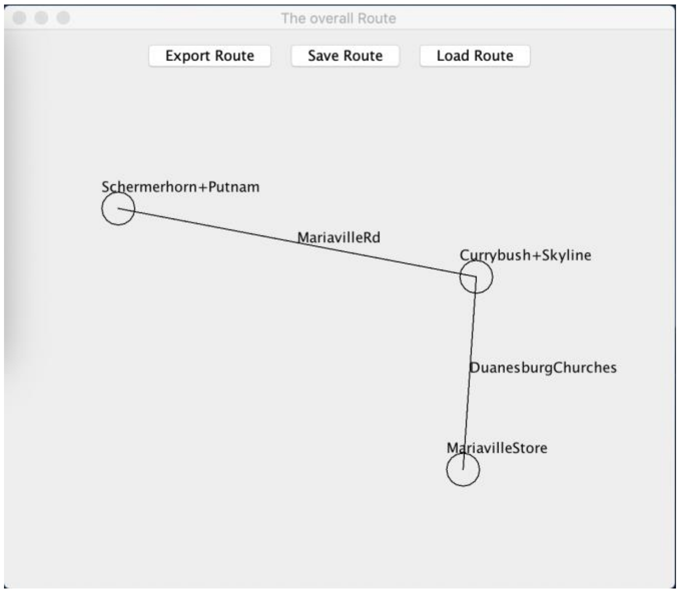
SegmentTest:

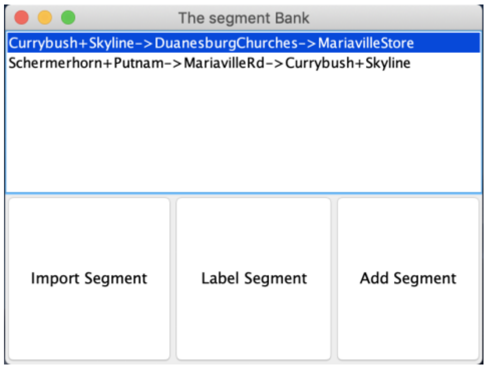
- testSegmentGetters
- testSegmentSetters
- testSameButDifferentPath
- testCloneMethod

IX. Results and Application Screenshots

Following are proposed features that we are able to successfully implement

- Able to import segments in CSV files
- Able to label segments in both the segmentBank and inside of the overall route display
- Able to add segments from the segmenBank to the overall route display
- Able to drag end-point around the canvas display
- Able to save and load routes, and export the overall route in CSV file format





Currybush+Skyline->DuanesburgChurches->MariavilleStore
Schermerhorn+Putnam->MariavilleRd->Currybush+Skyline

X. Lessons Learned

Throughout the first version of this project, our group has learned the following:

- There should be a clear initial understanding about the design concept and all the necessary features/roles before any coding
- As this was a group project, clear communication between teammates is fundamental for establishing a well-oiled team.
- By using Git for storing all the codes, every coder should understand the functionalities of Git, for avoiding any merge conflicts
- Learning Java Swing was not as smooth as our team anticipated.

XI. Version 2 Features

Version 1 of the cuesheet application has structured and flexible model and GUI which will allow for us to extend the features in version 2. There are several key updates we hope to implement. First of all, we would like to give users the ability to select the file format the final route is exported in. Current version only supports .csv file exports. Version 2 will allow the user to select between .csv and .txt file formats and potentially .pdf, if such implementation is not too time-consuming, given our time constraints.

We would also like users to be able to load multiple segments to our segment bank display. As of now, users have to load individual segment files into the segment bank display.

We hope to provide more capability to our canvas display so that intermediate segments from the canvas display can be added or remove. Presently, only beginning and ending segments can be added or removed from the canvas display.

On-the-canvas renaming is another GUI aesthetic upgrade we hope to implement optionally. In version 1, users can give nicknames to a segment's start, end, and intermediate pathname via the segment bank display. We would like to make the GUI fancier by allowing users to nickname start/end/intermediate parts of the route on the canvas itself.

Version 1 allows users to drag segment vertices on the canvas display. The segment vertex which is dragged is also highlighted in blue so that users can tell if a segment vertex is clicked and thus, draggable. An optional goal of version 2 application is to upgrade our GUI by showing the path of the dragged vertices. As of now, drag updates the canvas once the mouse is released. Implementing fancy drag will involve math which needs to dynamically calculate changes to not only segment vertices but also those of the paths/lines connecting segment vertices. Given the high level of complexity involved in this implementation, we are making this goal optional. Our focus throughout the entire project has been on the design of the app rather aesthetics of the GUI.

XII. Version 2 Timeline

Thursday (2/28): Select file formats to export + load multiple segments to bank
Tuesday (3/5): Add/Remove middle segment from route
Thursday (3/7): Fancy rename
Tuesday (3/12): Fancy Drag
Thursday (3/14): Final Presentation + final report