MetroMapMaker™
Software Design Description

Authors: Richard McKenna,
Mark Koshkin,
Debugging Enterprises™
Version 1.0

Abstract: This document that contains the software design MetroMapMaker™, a Java application for creating, modifying and saving custom metro maps.

0 Table of contents

1 Introduction

This is the Software Design Description for MetroMapMaker™. The format is based on IEEE Standard 1016-2009.

1.1 Purpose

This document provides a detailed blueprint for the MetroMapMaker applications. It includes UML class diagrams that explain the usage of packages, class relations, instance variables and methods.

1.2 Scope

MetroMapMaker™ is a software that is developed in Java programming language. However, the program is intended to be extended cross platform and also for the web use. The maps are stored using JSON files.

1.3 Definitions and Abbreviations

Class Diagram – a UML diagram that describes relationship between classes of the program, contains methods and static and instance variables

Framework – Collection of classes and interfaces that is used by an app for a specific problem.

IEEE – Institute of Electrical and Electronics Engineers

Java – A high-level multipurpose programming language developed by Oracle™

Metro – A subway system in a city

MMM – MetroMapMaker™

UML – Unified Modeling Language, a notation system for describing software structure

1.4 References

The MetroMapMaker™ Software Requirements Specification by Richard McKenna

1.5 Overview

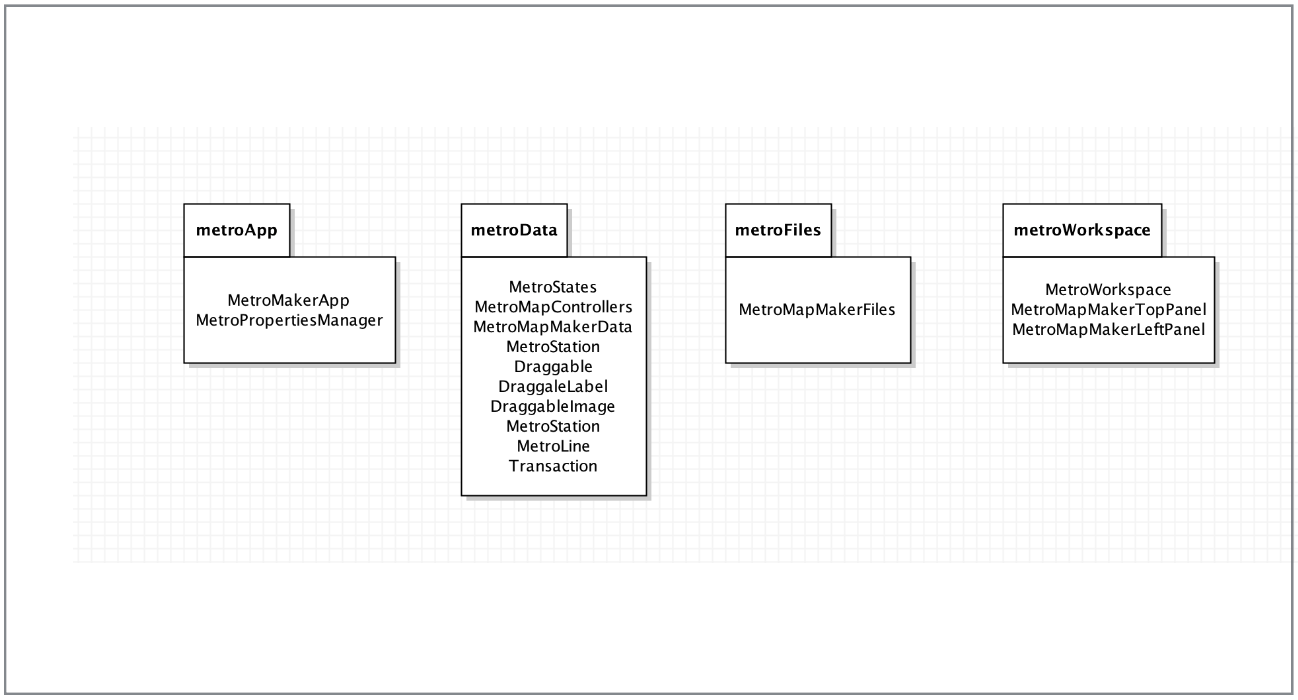
This Software Design Description is based on the MetroMapMaker™ Software Requirements Specification. It proposes solutions to the functionality defined by the Specification. Software Design Description explains the structure of the application.

2 Package-Level Design Viewpoint:

Four packages comprise the MetroMapMaker app. The design uses modular approach and each of the packages has a specific role. The packages communicate with each other in order to produce response to users input.

2.1 MetroMapMaker package overview

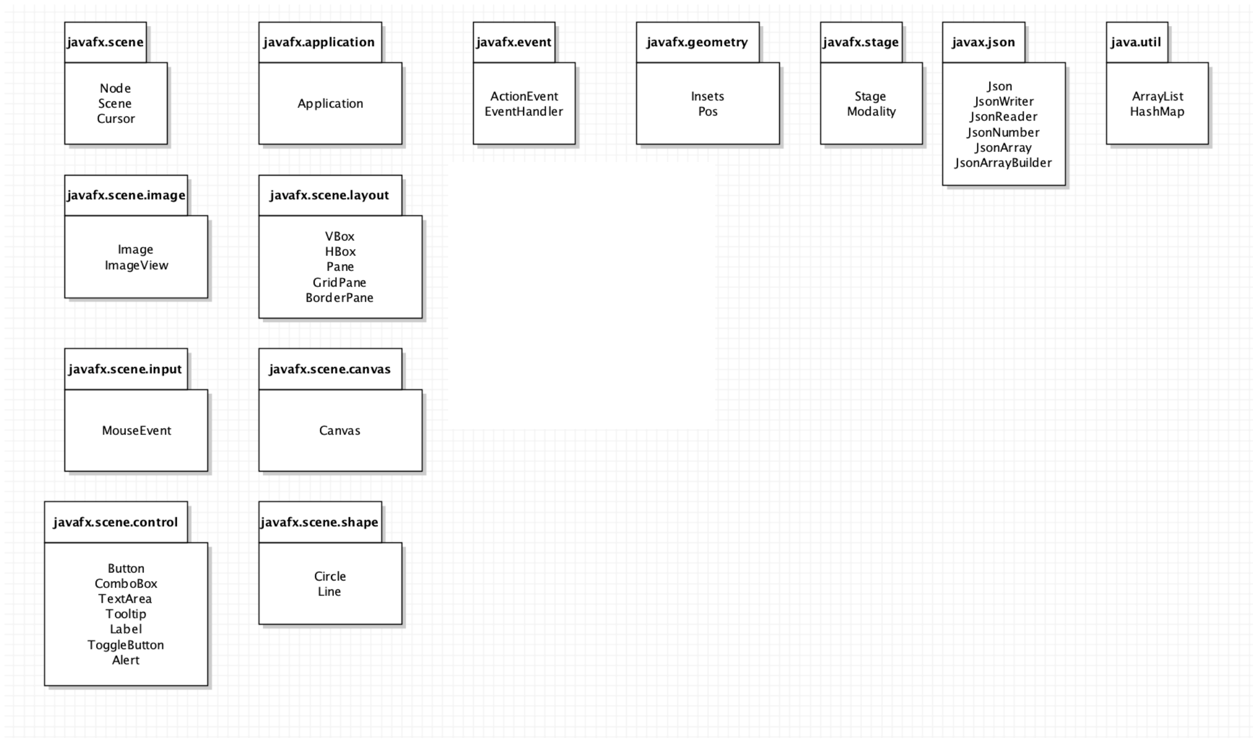
The following figure indicates the packages that were developed to accommodate MetroMapMaker



* Figure 2.1 Packages

2.2 Java API Usage:

Java is the primary platform for the MetroMapMaker. The chart specifies classes used.



* Figure 2.2: Java API Classes and Packages To Be Used

2.3 Java API Usage Descriptions

Table 2.31	**javafx.scene**
Node	Used for adding elements to panes
Scene	The main container for all the panels and layouts
Cursor	To change the appearance of cursor for different actions.

Table 2.32	**javafx.scene.control**
Button	Simple button for getting user input
ToggleButton	Button with multiple states
Tooltip	Verbal explanation of what button does
Label	The labels for sections of panel
ComboBox	Choosing lines or metro stations
TextArea	Getting text input
Alert	Informing user or getting input

Table 2.33	**javafx.scene.image**
Image	Displayable image object
ImageView	For displaying objects

Table 2.34	**javafx.scene.input**
MouseEvent	For user mouse clicks and drags

Table 2.35	**javafx.scene.canvas**
Canvas	All the drawing happens here

Table 2.36	**javafx.scene.shape**
Circle	For metro stations
Line	For metro lines

Table 2.37	**javafx.scene.layout**
HBox	For file panel and mini panels
VBox	For left panel
GridPane	For buttons on the panels
BorderPane	The main pane that holds the rest

Table 2.38	**javafx.stage**
Stage	The main window of the program
Modality	The priority of windows

Table 2.39	**javafx.event**
EventHandler	Responding to user events
ActionEvent	Responding to user events

Table 2.310	**javafx.geometry**
Insets	Paddings between buttons
Pos	A set of values for describing vertical and horizontal positioning and alignment.

Table 2.311	**java.util**
ArrayList	To store lines, stations, shapes
HashMap	For storing properties

Table 2.312	javax.json
Json	Saves data from the app into a convenient and universal form
JsonArray	Creates json arrays
JsonReader	Reads form json file
JsonWriter	Writes into json file
JsonNumber	For operating numbers within json
JsonBuilder	Builds the json file

Table 2.313	javafx.application
Application	MetroMapMaker extends this class, as it is an application

3 Class-Level Design Viewpoint

The following diagram describes the interactions and relationships between classes. It builds the skeleton for the development of MetroMapMaker. The role of this objects is respond to users actions through interaction and exchange of data.

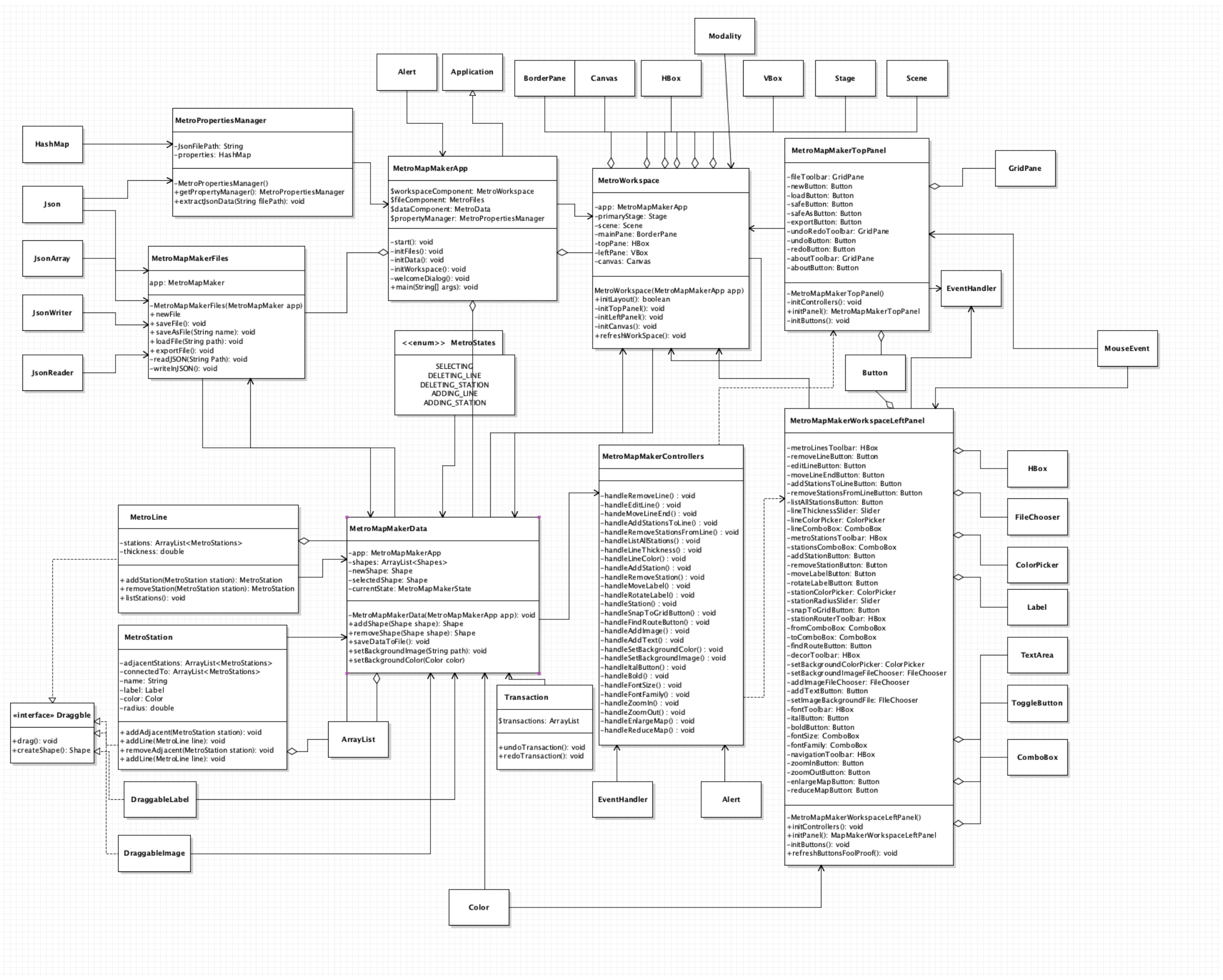

Modality
Alert
Application
BorderPane
Canvas
HBox
VBox
Stage
Scene
HashMap
Json
JsonArray
JsonWriter
JsonReader
MetroPropertiesManager
-JsonFilePath: String
-properties: HashMap
-MetroPropertiesManager()
+getPropertyManager(): MetroPropertiesManager
+extractJsonData(String filePath): void
MetroMapMakerApp
$workspaceComponent: MetroWorkspace
$fileComponent: MetroFiles
$dataComponent: MetroData
$propertyManager: MetroPropertiesManager
-start(): void
-initFiles(): void
-initData(): void
-initWorkspace(): void
-welcomeDialog(): void
+main(String[] args): void
MetroMapMakerFiles
app: MetroMapMaker
-MetroMapMakerFiles(MetroMapMaker app)
+newFile
+saveFile(): void
+saveAsFile(String name): void
+loadFile(String path): void
+exportFile(): void
-readJSON(String Path): void
-writeInJSON(): void
<<enum>> MetroStates
SELECTING
DELETING_LINE
DELETING_STATION
ADDING_LINE
ADDING_STATION
MetroWorkspace
-app: MetroMapMakerApp
-primaryStage: Stage
-scene: Scene
-mainPane: BorderPane
-topPane: HBox
-leftPane: VBox
-canvas: Canvas
MetroWorkspace(MetroMapMakerApp app)
+initLayout(): boolean
-initTopPanel(): void
-initLeftPanel(): void
-initCanvas(): void
+refreshWorkSpace(): void
MetroMapMakerTopPanel
-fileToolbar: GridPane
-newButton: Button
-loadButton: Button
-safeButton: Button
-safeAsButton: Button
-exportButton: Button
-undoRedoToolbar: GridPane
-undoButton: Button
-redoButton: Button
-aboutToolbar: GridPane
-aboutButton: Button
-MetroMapMakerTopPanel()
-initControllers(): void
+initPanel(): MetroMapMakerTopPanel
-initButtons(): void
GridPane
EventHandler
MouseEvent
Button
MetroMapMakerWorkspaceLeftPanel
-metroLinesToolbar: HBox
-removeLineButton: Button
-editLineButton: Button
-moveLineEndButton: Button
-addStationsToLineButton: Button
-removeStationsFromLineButton: Button
-listAllStationsButton: Button
-lineThicknessSlider: Slider
-lineColorPicker: ColorPicker
-lineComboBox: ComboBox
-metroStationsToolbar: HBox
-stationsComboBox: ComboBox
-addStationButton: Button
-removeStationButton: Button
-moveLabelButton: Button
-rotateLabelButton: Button
-stationColorPicker: ColorPicker
-stationRadiusSlider: Slider
-snapToGridButton: Button
-stationRouterToolbar: HBox
-fromComboBox: ComboBox
-toComboBox: ComboBox
-findRouteButton: Button
-decorToolbar: HBox
-setBackgroundColorPicker: ColorPicker
-setBackgroundImageFileChooser: FileChooser
-addImageFileChooser: FileChooser
-addTextButton: Button
-setImageBackgroundFile: FIleChooser
-fontToolbar: HBox
-italButton: Button
-boldButton: Button
-fontSize: ComboBox
-fontFamily: ComboBox
-navigationToolbar: HBox
-zoomInButton: Button
-zoomOutButton: Button
-enlargeMapButton: Button
-reduceMapButton: Button
-MetroMapMakerWorkspaceLeftPanel()
+initControllers(): void
+initPanel(): MapMakerWorkspaceLeftPanel
-initButtons(): void
+refreshButtonsFoolProof(): void
HBox
FileChooser
ColorPicker
Label
TextArea
ToggleButton
ComboBox
MetroMapMakerControllers
-handleRemoveLine() : void
-handleEditLine() : void
-handeMoveLineEnd() : void
-handleAddStationsToLine() : void
-handleRemoveStationsFromLine() : void
-handleListAllStations() : void
-handleLineThickness() : void
-handleLineColor() : void
-handleAddStation() : void
-handleRemoveStation() : void
-handleMoveLabel() : void
-handleRotateLabel() : void
-handleStation() : void
-handleSnapToGridButton() : void
-handleFindRouteButton() : void
-handleAddImage() : void
-handleAddText() : void
-handleSetBackgroundColor() : void
-handleSetBackgroundImage() : void
-handleItalButton() : void
-handleBold() : void
-handleFontSize() : void
-handleFontFamily() : void
-handleZoomIn() : void
-handleZoomOut() : void
-handleEnlargeMap() : void
-handleReduceMap() : void
EventHandler
Alert
MetroLine
-stations: ArrayList<MetroStations>
-thickness: double
+addStation(MetroStation station): MetroStation
+removeStation(MetroStation station): MetroStation
+listStations(): void
MetroMapMakerData
-app: MetroMapMakerApp
-shapes: ArrayList<Shapes>
-newShape: Shape
-selectedShape: Shape
-currentState: MetroMapMakerState
-MetroMapMakerData(MetroMapMakerApp app): void
+addShape(Shape shape): Shape
+removeShape(Shape shape): Shape
+saveDataToFile(): void
+setBackgroundImage(String path): void
+setBackgroundColor(Color color)
MetroStation
-adjacentStations: ArrayList<MetroStations>
-connectedTo: ArrayList<MetroStations>
-name: String
-label: Label
-color: Color
-radius: double
+addAdjacent(MetroStation station): void
+addLine(MetroLine line): void
+removeAdjacent(MetroStation station): void
+addLine(MetroLine line): void
«interface» Draggble
+drag(): void
+createShape(): Shape
ArrayList
Transaction
$transactions: ArrayList
+undoTransaction(): void
+redoTransaction(): void
DraggableLabel
DraggableImage
Color

*Figure 3.1: UML Class Diagram

4 Method-Level Design Viewpoint

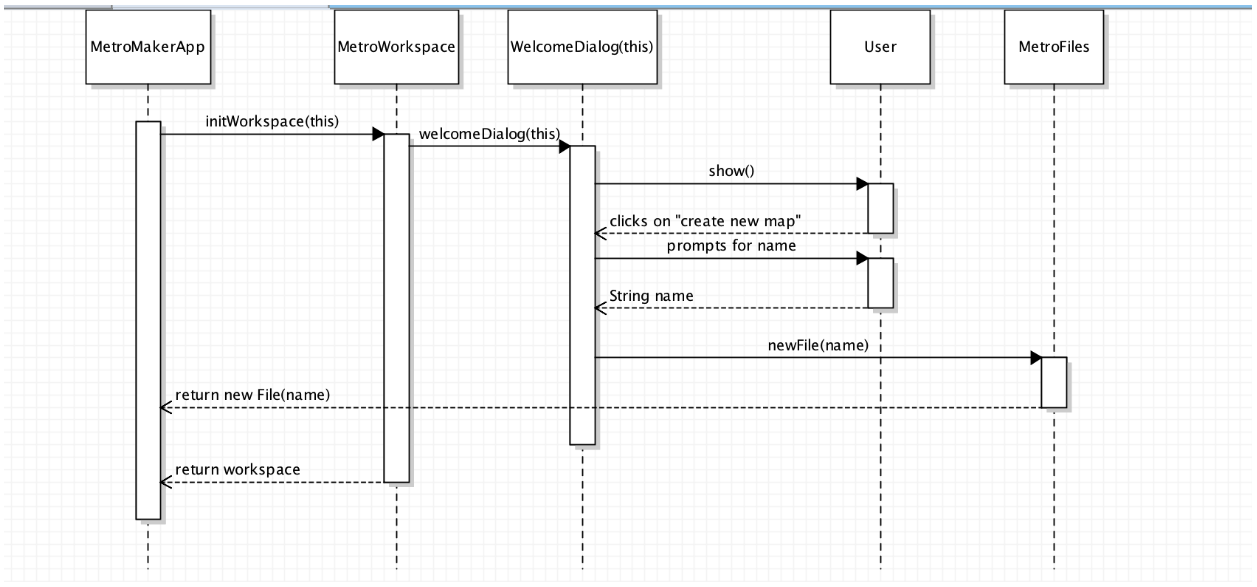


Figure 4.1: New Map from WelcomeDialog

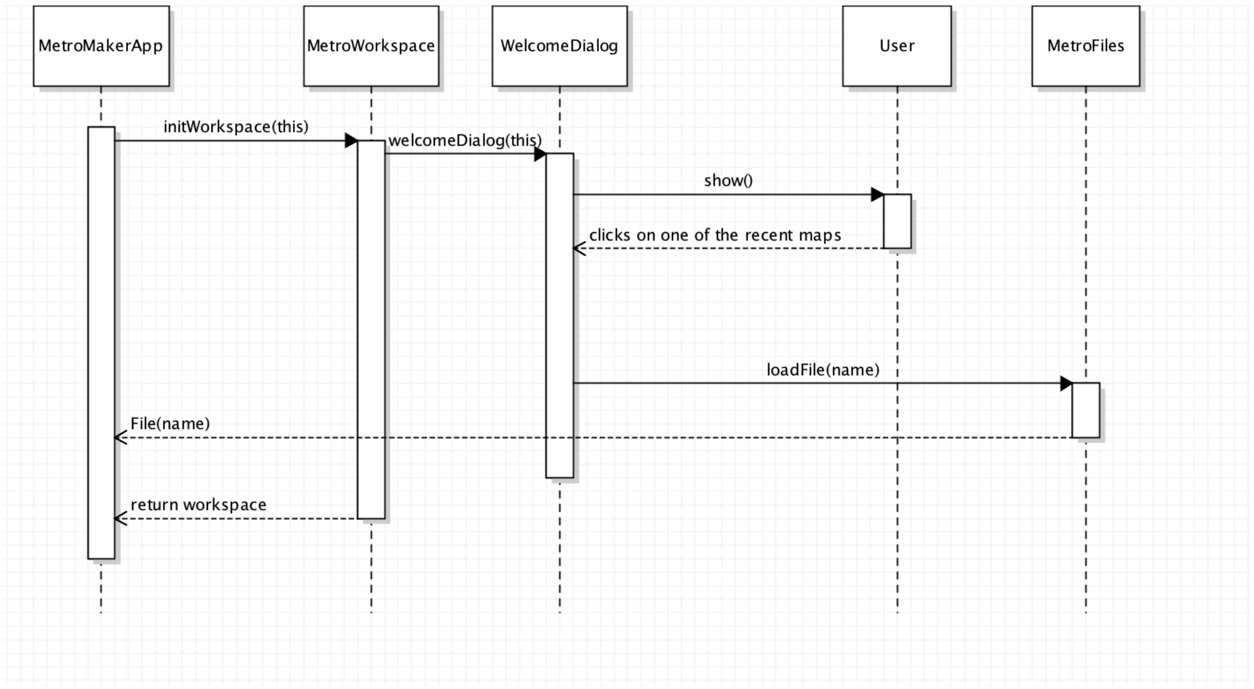


*Figure 4.2: Load Recent Map Welcome Dialog

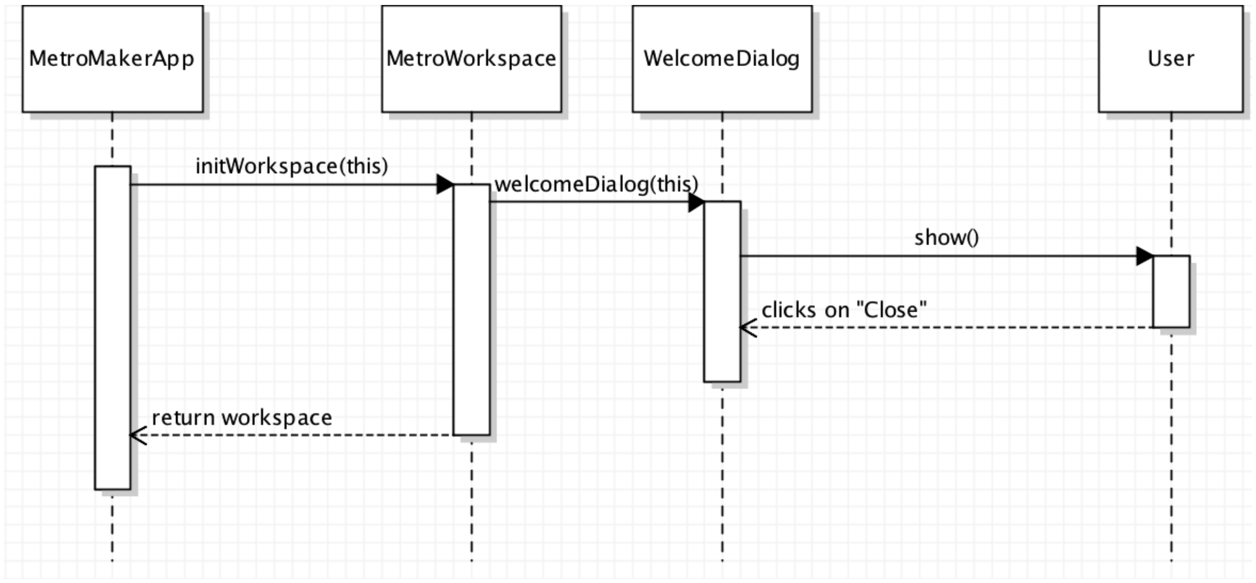


*Figure 4.3: Close Welcome Dialog

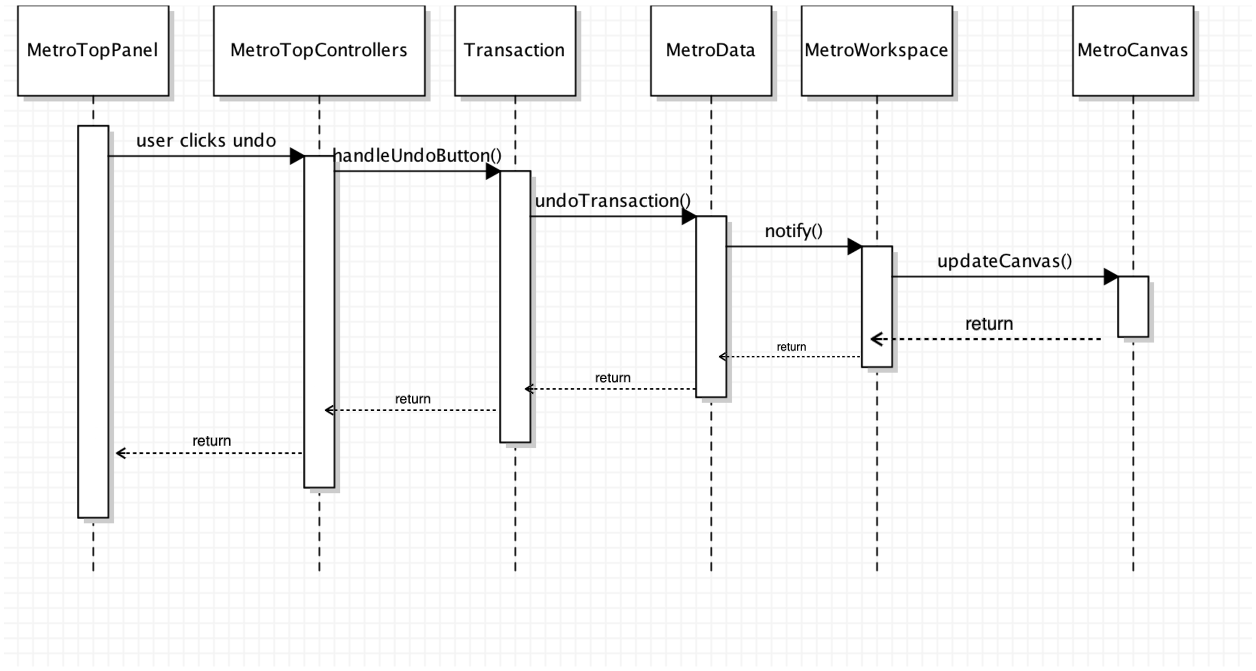


*Figure 4.9: Undo Transaction

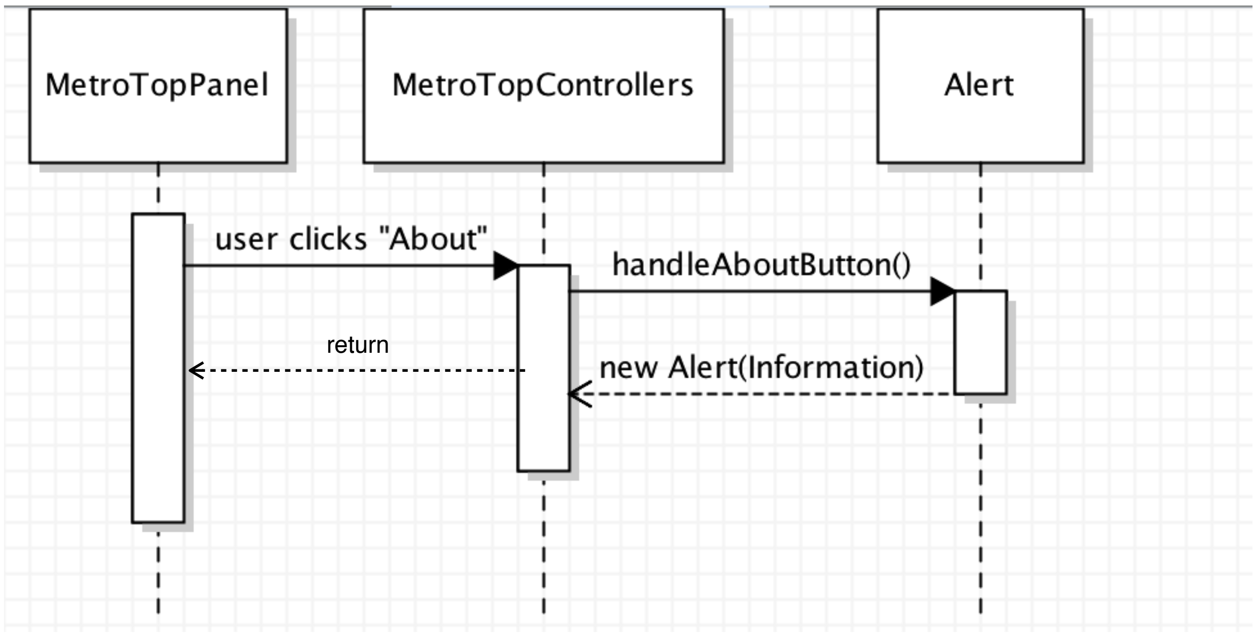


*Figure 4.11: About Dialog

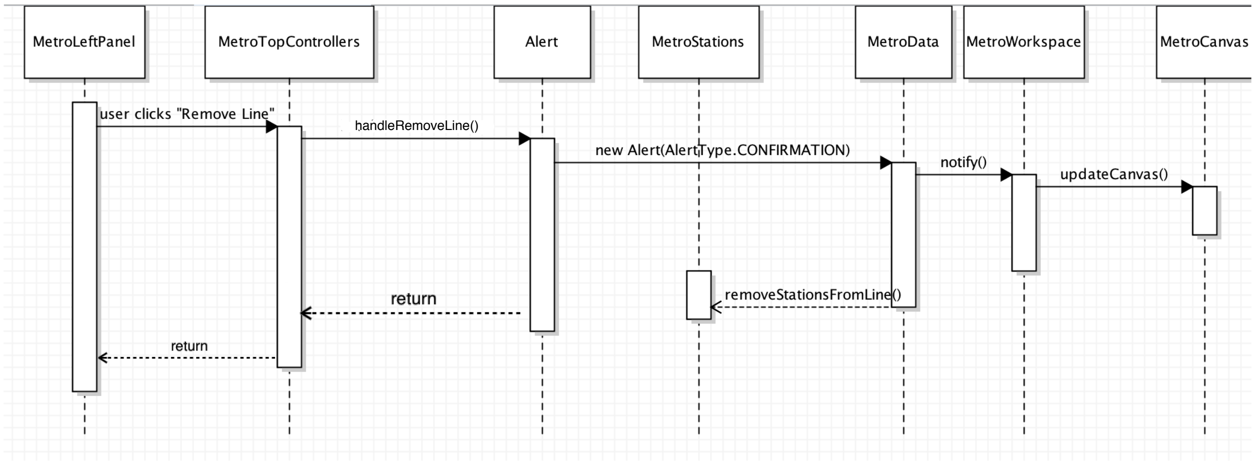


*Figure 4.13: Remove Line

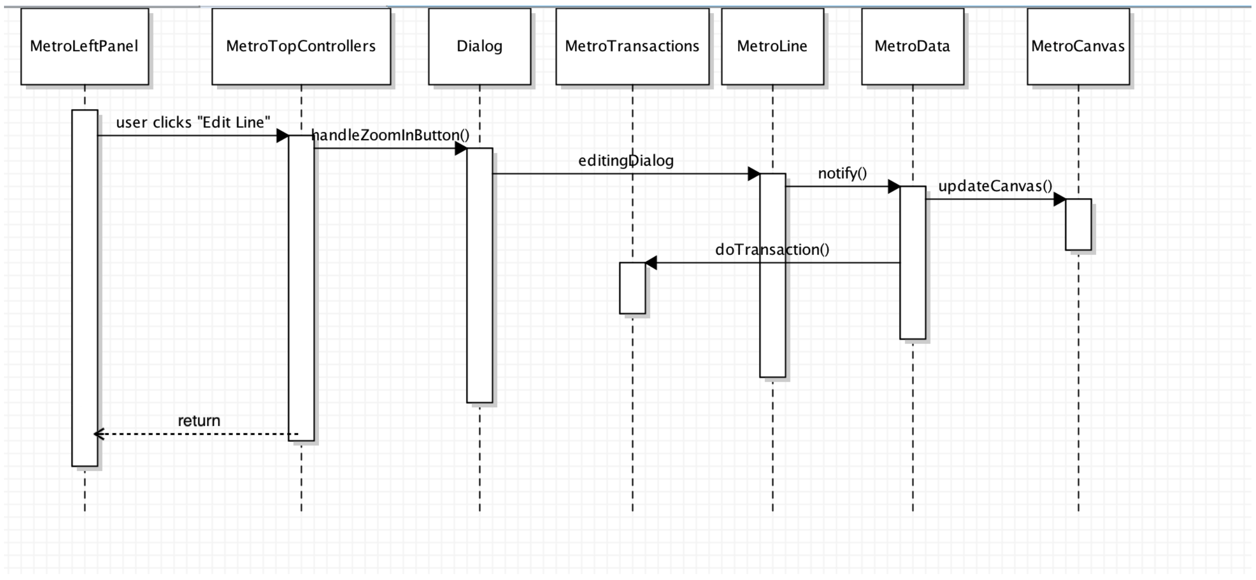


*Figure 4.14: Edit Line

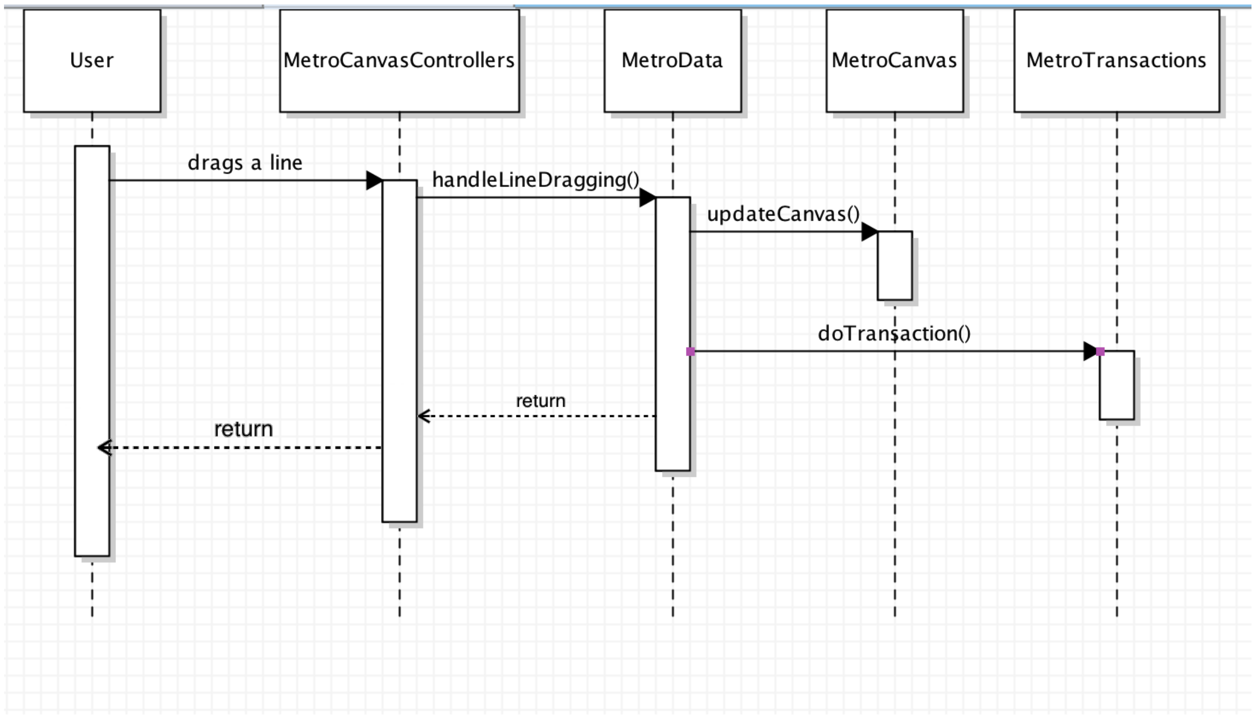


*Figure 4.15: Move Line End

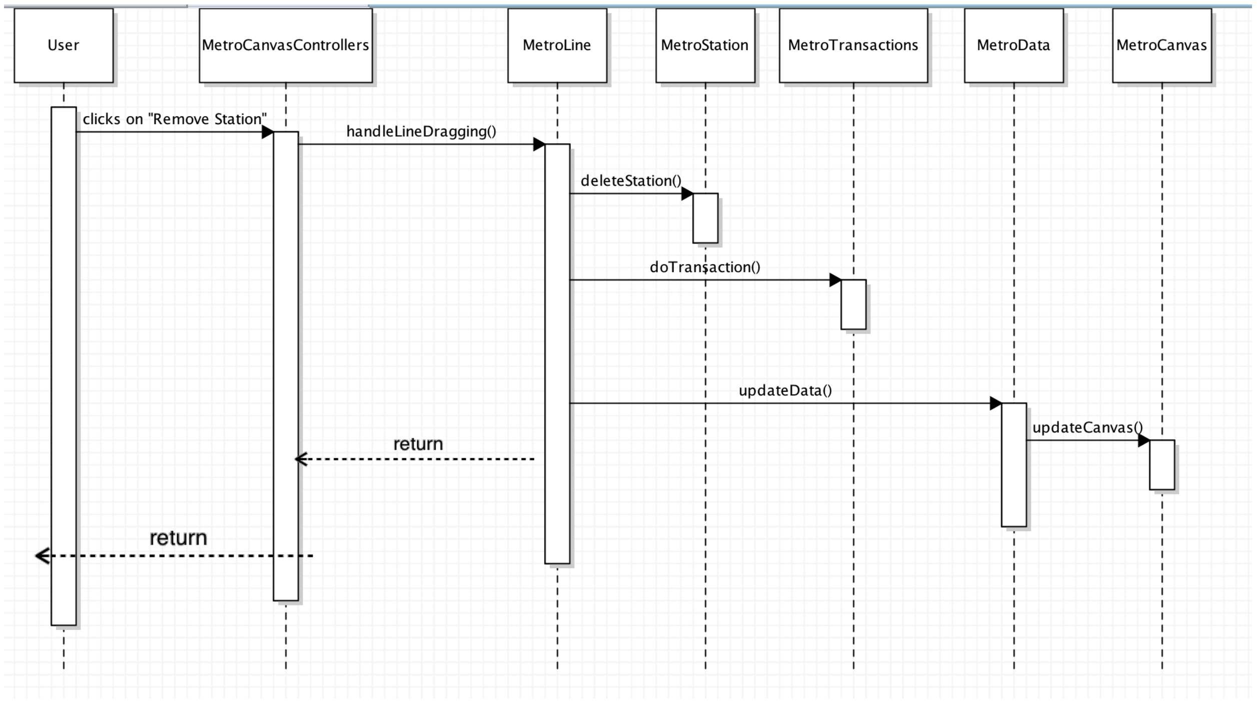


*Figure 4.17: Remove Stations from the Line

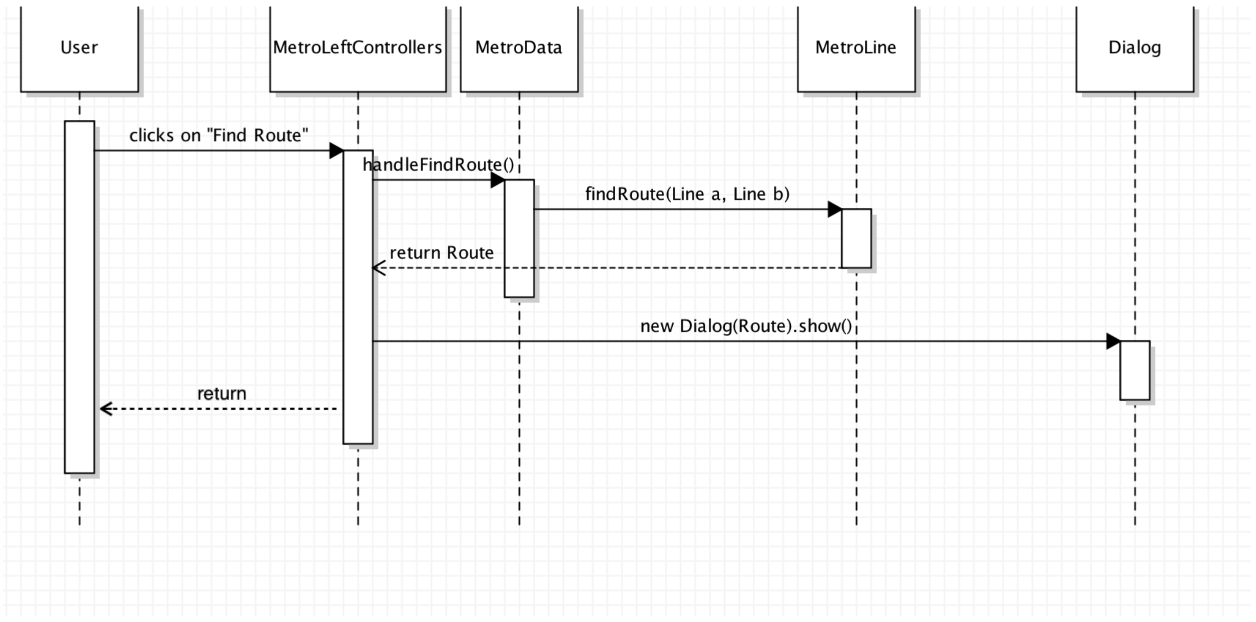


*Figure 4.23: Find Route

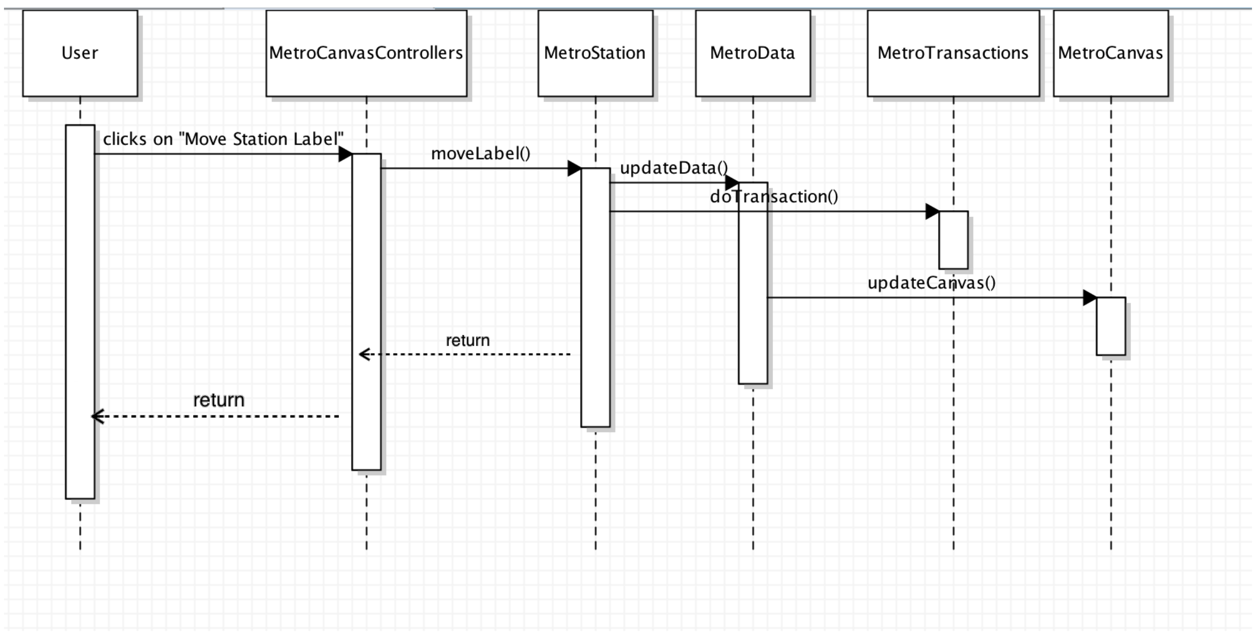


*Figure 4.23: Move Station Label

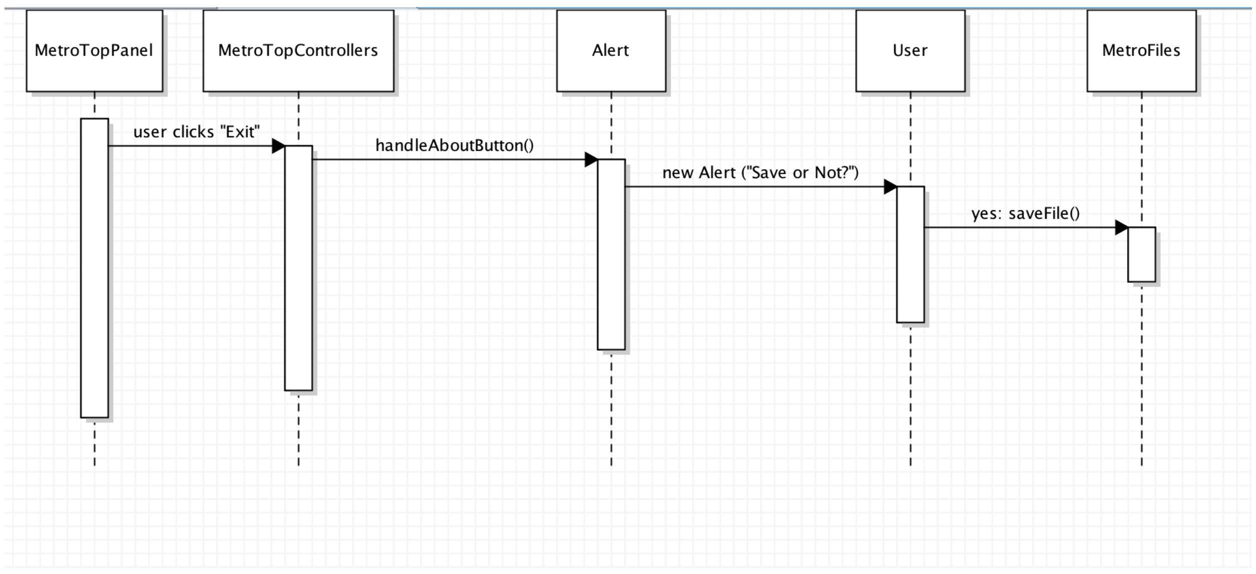


*Figure 4.39: Exit

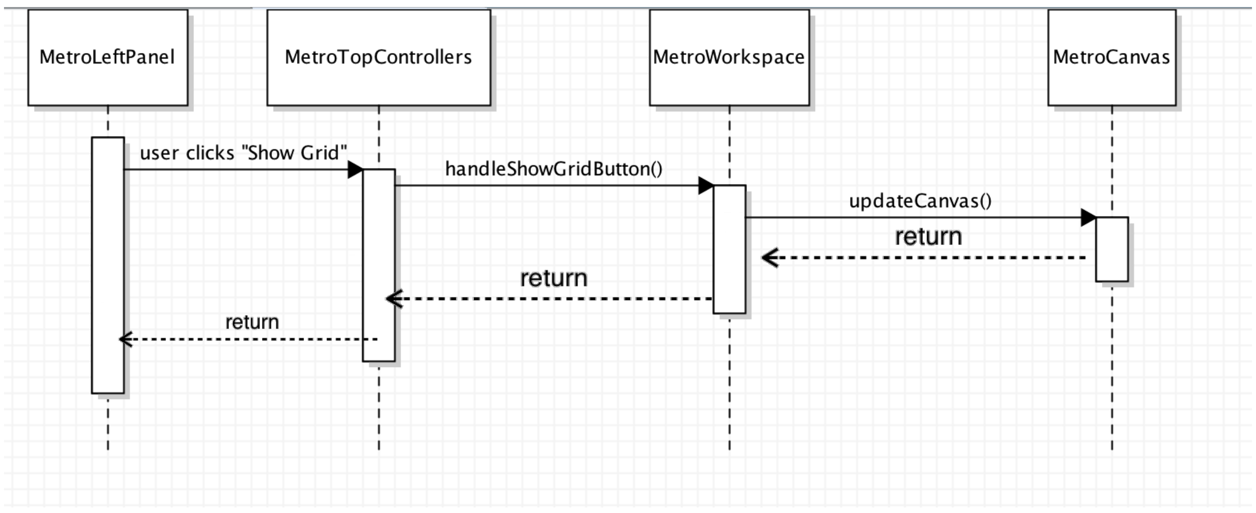


*Figure 4.36: Toggle Grid

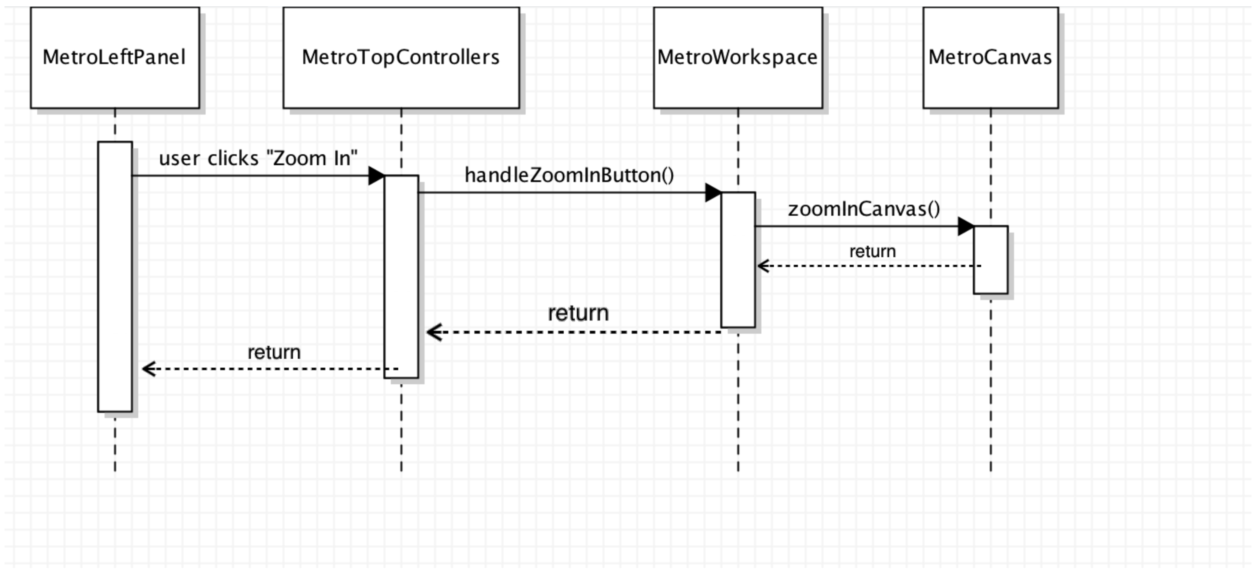


*Figure 4.37: Zoom In

5 File Structure

The file system of MetroMapMaker has the following structure. It is responsible for exporting of created maps, as well as construction of GUI.

- Main Directory/
    - javax.json-1.0.4.jar (for parsing json objects)
    - properties.json
    - scr/
        - data/
            - metroStyle.css
            - melcomeDialogue.css
        - images/
            - icons and logos
        - export/
            - where files are saved by default
        - java source code …

properties.json is the file that contains string constants, icon locations, tooltips, and it is placed in the main directory so that all the packages and components can have access to it. There is a separate folders for graphic content as well as the css data. The default export folder is provided.

6 Architecture Discussion

6.1 Design Choice

The described software design aim to put modularity in a good use. The main point is to keep different components independent from each other, but still allow them to interact. A positive aspect of such implementation is that things are kept in one place. For example, all the style settings are kept separately in css files away from java code. Inline styling would really affect the readability of the code.

There are drawbacks in my design. For example, some classes are "overcrowded": MetroLeftPanel Class contains over 50 GUI variable definitions. There is also a lot of interactions between all the classes and it is hard to keep track of them. An example of such obstacle is that every kind of transaction needs its own implementation. Undo/Redo handles would have to be aware.

6.2 Interaction of the components

The components exchange data with each other, whenever user provides input. This was shown in the UML sequence digram section. For example, when user presses save file button, the button fires an event, that is handled by a controller class. In turn, the controller class communicates to data class, telling it to call save file function, which is a part of file system data object. The data object is processed by the MetroFile class, and the file is written to the hard disk.