The Metro Map Maker
Software Design Description

Author: Brendan Kondracki
October 2017

Abstract: This document describes the software design for The Metro Map Maker, an application to create and export custom subway maps.

1 Introduction

This is the Software Design Description (SDD) for The Metro Map Maker application. This document is based on the IEEE Standard 1016-2009 recommendation for software design.

1.1 Purpose

This document is the blueprint for the creation of The Metro Map Maker application. This design will use UML class diagrams to provide details with regards to all packages, classes, instance variables, class variables, and method signatures required to build the application. UML sequence diagrams will also be used to describe the procedures for handling user input/interaction.

1.2 Scope

The Metro Map Maker will be designed as a standalone application, requiring no resources from outside applications to function properly. The application will utilize the Desktop Java Framework in order to process user actions regarding the loading and selection of files, as well as the jTPS framework to handle undo/redo requests. In addition, the Properties Manger framework will be used to handle the loading of data from an xml file into the application. The use of these three frameworks was decided to be the most beneficial approach to designing this application, over the use of newly created frameworks/classes due to their ability to effectively design a workspace style application and implement key features such as undo/redo. This application will be designed using the Java programming language.

1.3 Definitions, acronyms, and abbreviations

Class Diagram – A UML document format that describes classes graphically. Specifically, it describes their instance variables, method headers, and relationships to other classes.

IEEE – Institute of Electrical and Electronics Engineers, the "world's largest professional association for the advancement of technology".

Framework – In and object-oriented language, a collection of classes and interfaces that collectively provide a service for building applications or additional frameworks all with a common need.

Java – A high-level programming language that uses virtual machine layer between the Java application and the hardware to provide program portability.

Metro – A subway system in a city.

Sequence Diagram – A UML document format that specifies how object methods interact with one another.

Subway - An underground electric railroad.

UML - Unified Modeling Language, a standard set of document formats for designing software graphically.

1.4 References

IEEE Std 830™ -1998 (R2009) – IEEE Standard for Information Technology – Systems Design – Software Design Descriptions.

1.5 Overview

This SDD document provides the overall design for the construction of The Metro Map Maker application. Section 2 of this document will contain the Package-Level Viewpoint, displaying both a graphical representation, and a textual description of the packages and frameworks to be used in this application. Section 3 will then describe the Class-Level Viewpoint, presenting the overall function and design of the classes to be used for the application. UML Class Diagrams will be utilized to specify this. Section 4 will provide the Method-Level System Viewpoint, giving an overview as to how methods will be used with respect to one another. Section 5 will describe deployment information such as file structures and formats to use. Finally, section 6 will provide a Table of Contents, Index, and references. The UML Diagrams displayed in this document were created using the VioletUML editor.

2 Package-Level Design Viewpoint

As stated before, the design for this application will utilize the Desktop Java Framework, the Properties Manager Framework, and the jTPS Framework in addition to The Metro Map Maker application. Below are descriptions for these components, as well as the Java API which will be used to build them.

2.1 Metro Map Maker Overview

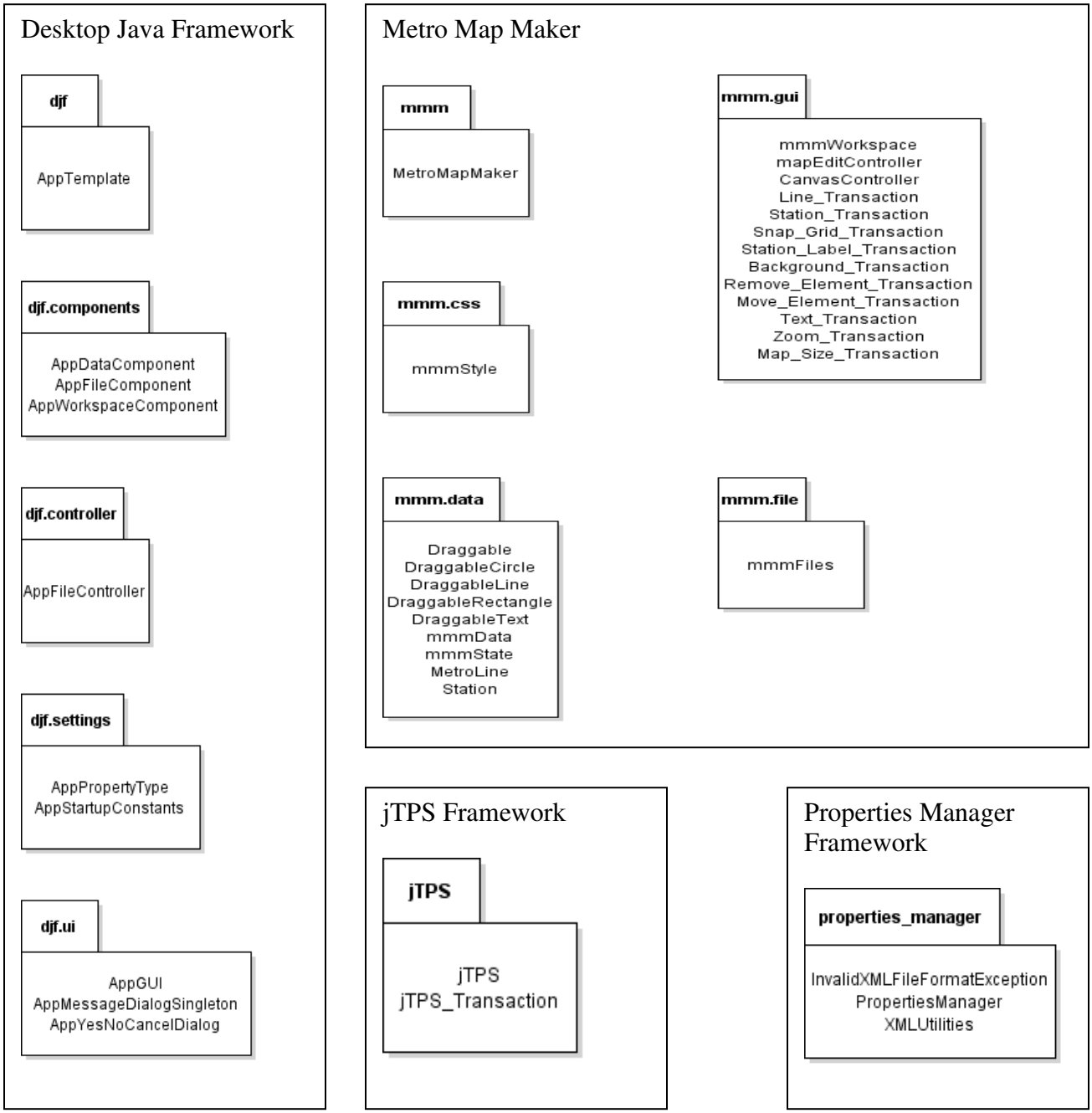


Figure 2.1: Design Packages Overview

2.2 Java API Usage

The frameworks and The Metro Map Maker application will make use of the following Java classes, specified below.

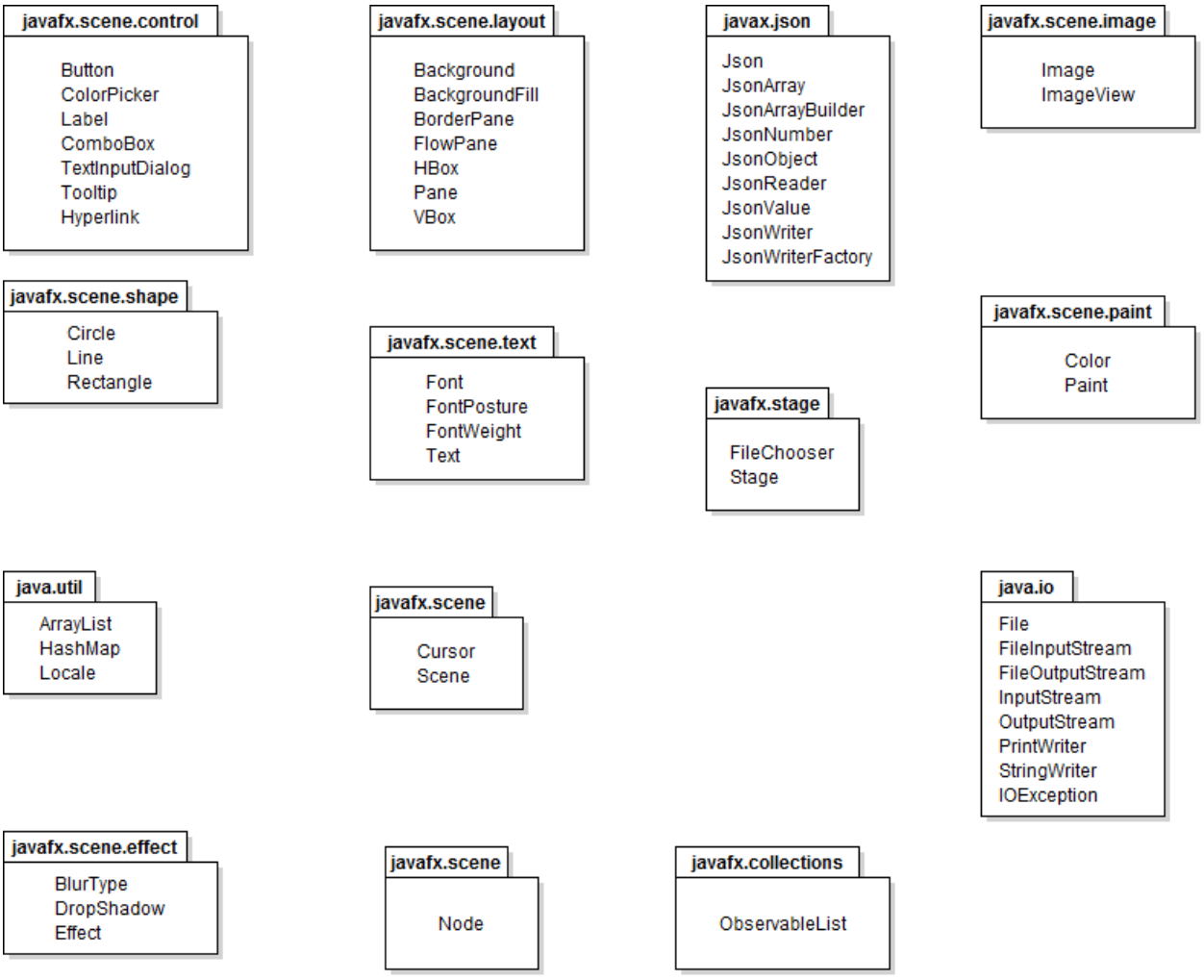


Figure 2.2: Java API Classes and Packages To Be Used

2.3 Java API Usage Descriptions

Tables 2.1-2.14 below summarize how each of these classes will be used

Class/Interface	Use
Button	For adding buttons to the application
ColorPicker	For changing the color of lines, background, etc.
Label	For adding labels to stations
ComboBox	For selecting stations from list
TextInputDialog	For changing name of station
Tooltip	For adding tooltips to buttons
Hyperlink	For user selection of recently edited maps in the welcome dialog

Table 2.1: Uses for classes in the Java API's javafx.scene.control package

Class/Interface	Use
Background	For creating application background
BackgroundFill	For adding fill color to application's background
BorderPane	For creating application's pane
FlowPane	For creating application's file toolbars
HBox	For creating workspace editing toolbars
VBox	For holding the workspace's editing toolbars

Table 2.2: Uses for classes in the Java API's javafx.scene.layout package

Class/Interface	Use
Json	For creating Json file to save application's data
JsonArray	For storing necessary application data
JsonArrayBuilder	For constructing JsonArray
JsonNumber	For storing integer or double value into the JsonArray
JsonObject	For storing multiple objects into a single Json object
JsonReader	For reading data from a specified Json file
JsonValue	For retrieving data from a specified key
JsonWriter	For outputting necessary data to a Json file
JsonWriterFactory	For creating JsonWriter instances

Table 2.3: Uses for classes in the Java API's javax.json package

Class/Interface	Use
Image	For creating objects for loaded in images
ImageView	For displaying loaded in images on the canvas

Table 2.4: Uses for classes in the Java API's javafx.scene.image package

Class/Interface	Use
Circle	For adding station circles to the canvas
Line	For adding lines to the canvas
Rectangle	For adding draggable images to the canvas

Table 2.5: Uses for classes in the Java API's javafx.scene.shape package

Class/Interface	Use
Font	For changing the font of a station's name
FontPosture	For italicizing a station's name
FontWeight	For bolding a station's name
Text	For creating a text variable for a station's name

Table 2.6: Uses for classes in the Java API's javafx.scene.text package

Class/Interface	Use
FileChooser	For selecting Metro Map to be loaded into the application
Stage	For creating new stage for the application

Table 2.7: Uses for classes in the Java API's javafx.stage package

Class/Interface	Use
Color	For creating color variable for selected color from color picker
Paint	For obtaining paint value of selected color

Table 2.8: Uses for classes in the Java API's javafx.scene.paint package

Class/Interface	Use
ArrayList	For storing stations on a given line
HashMap	For storing values from created JsonArray
Locale	For representing region (US)

Table 2.9: Uses for classes in the Java API's java.util package

Class/Interface	Use
Cursor	For changing style of cursor based on state of the application
Scene	For creating scene of the application

Table 2.10: Uses for classes in the Java API's javafx.scene package

Class/Interface	Use
File	For storing selected file into a variable
FileInputStream	For loading in data from a selected Json file
FileOutputStream	For outputting data into a Json file
InputStream	The apparent type of a FileInputStream object
OutputStream	The apparent type of a FileOutputStream object
PrintWriter	For printing formatted Json file to a text-output stream
StringWriter	For creating required JsonWriter
IOException	Exception thrown for failed or interrupted I/O operations

Table 2.11: Uses for classes in the Java API's java.io package

Class/Interface	Use
BlurType	For softening a shadow effect on highlighted object
DropShadow	For rendering a shadow behind selected object
Effect	For setting highlighted effect on selected object

Table 2.12: Uses for classes in the Java API's javafx.scene.effect package

Class/Interface	Use
Node	Base class for scene graph nodes

Table 2.13: Uses for classes in the Java API's javafx.scene package

Class/Interface	Use
ObservableList	List which allows listeners to track changes when they occur. Will be used to store all elements added to the canvas.

Table 2.14 Uses for classes in the Java API's javafx.collections package

3 Class-Level Design Viewpoint

The following UML class diagrams are used to represent the classes used in the creation of the application. The diagrams will start with a general overview, and proceed down to more detailed diagrams.

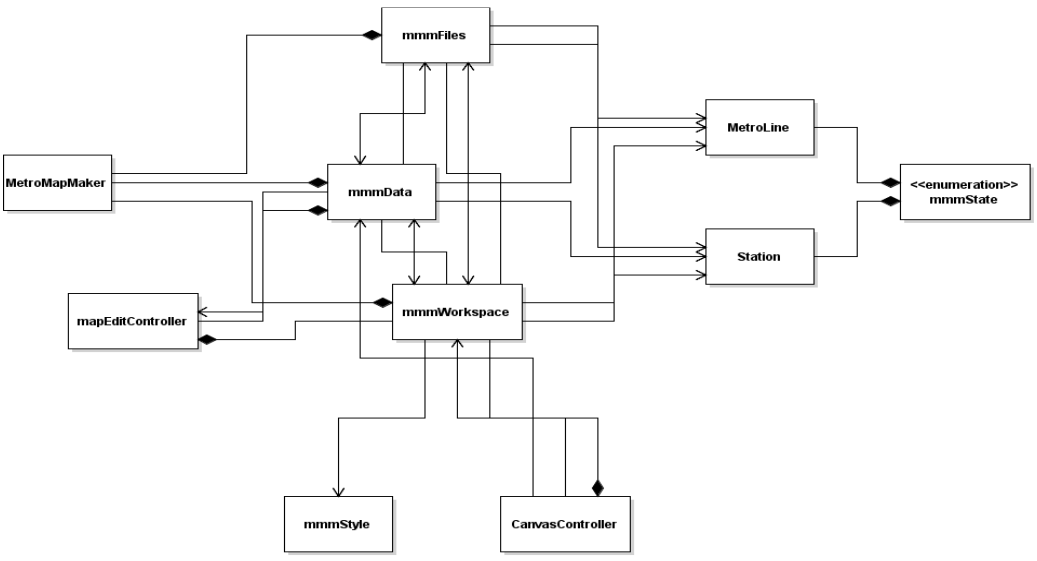


Figure 3.1: Metro Map Maker Overview UML Diagram

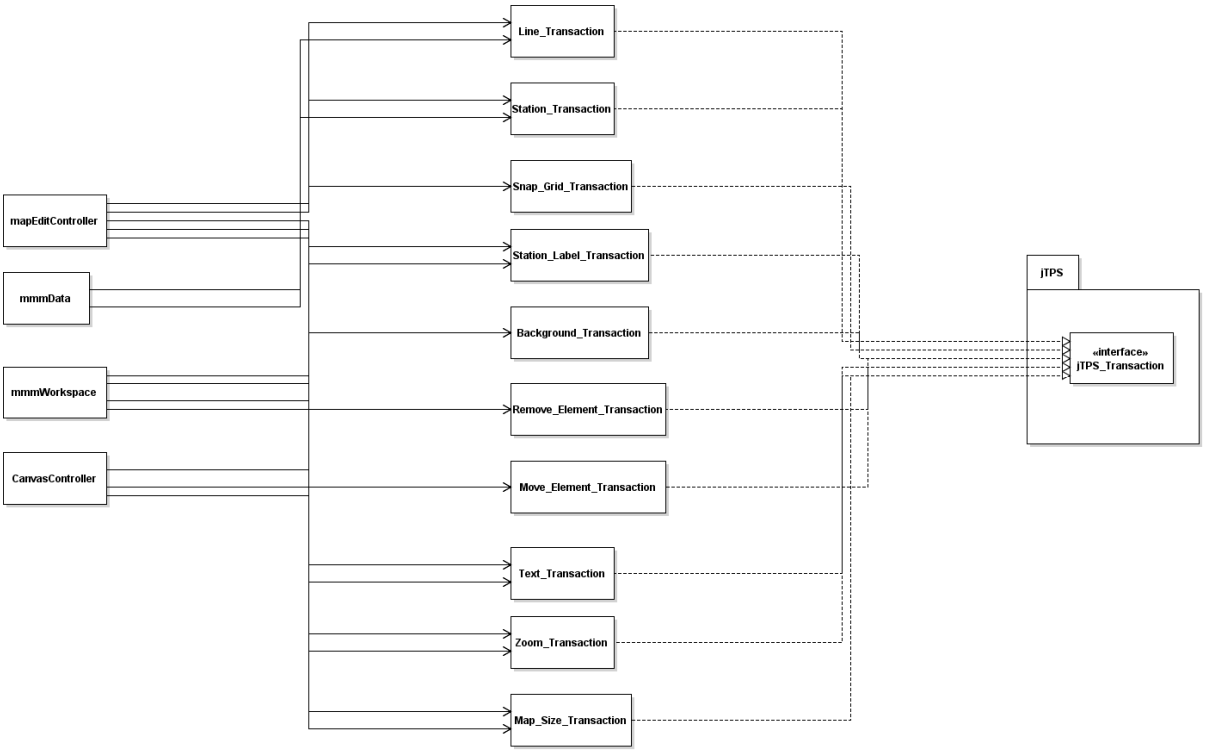


Figure 3.2: Transactions Overview UML Diagram

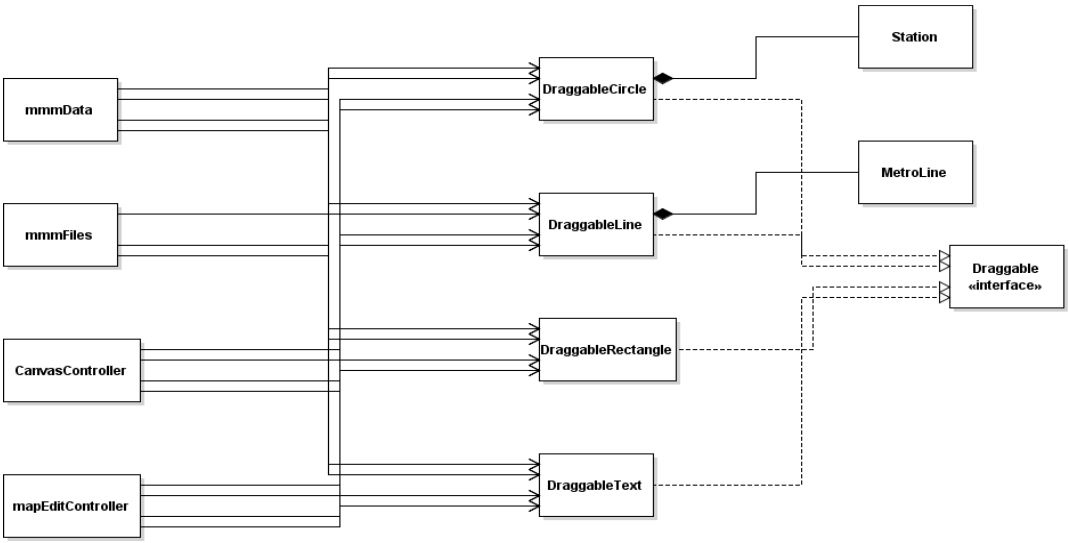


Figure 3.3: Draggable Overview UML Diagram

Figure 3.4: Detailed MetroMapMaker, mmmFiles, and mmmData UML Class Diagrams

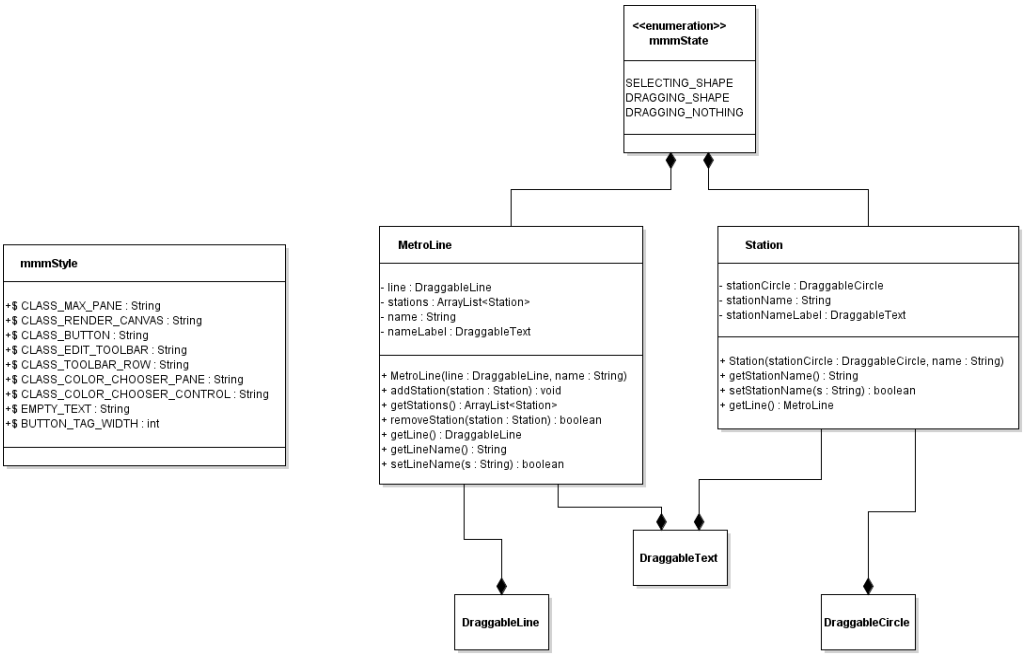


Figure 3.5: Detailed MetroLine, Station, mmmStyle, and mmmState UML Class Diagrams

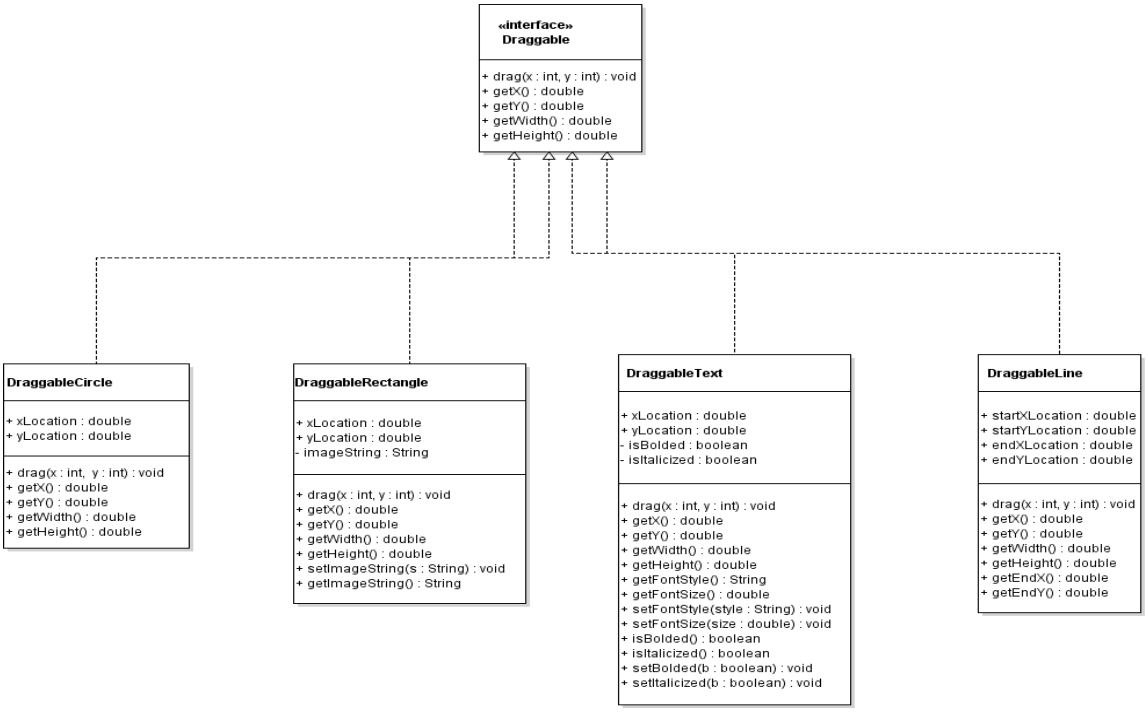


Figure 3.6: Detailed Draggable UML Class Diagrams

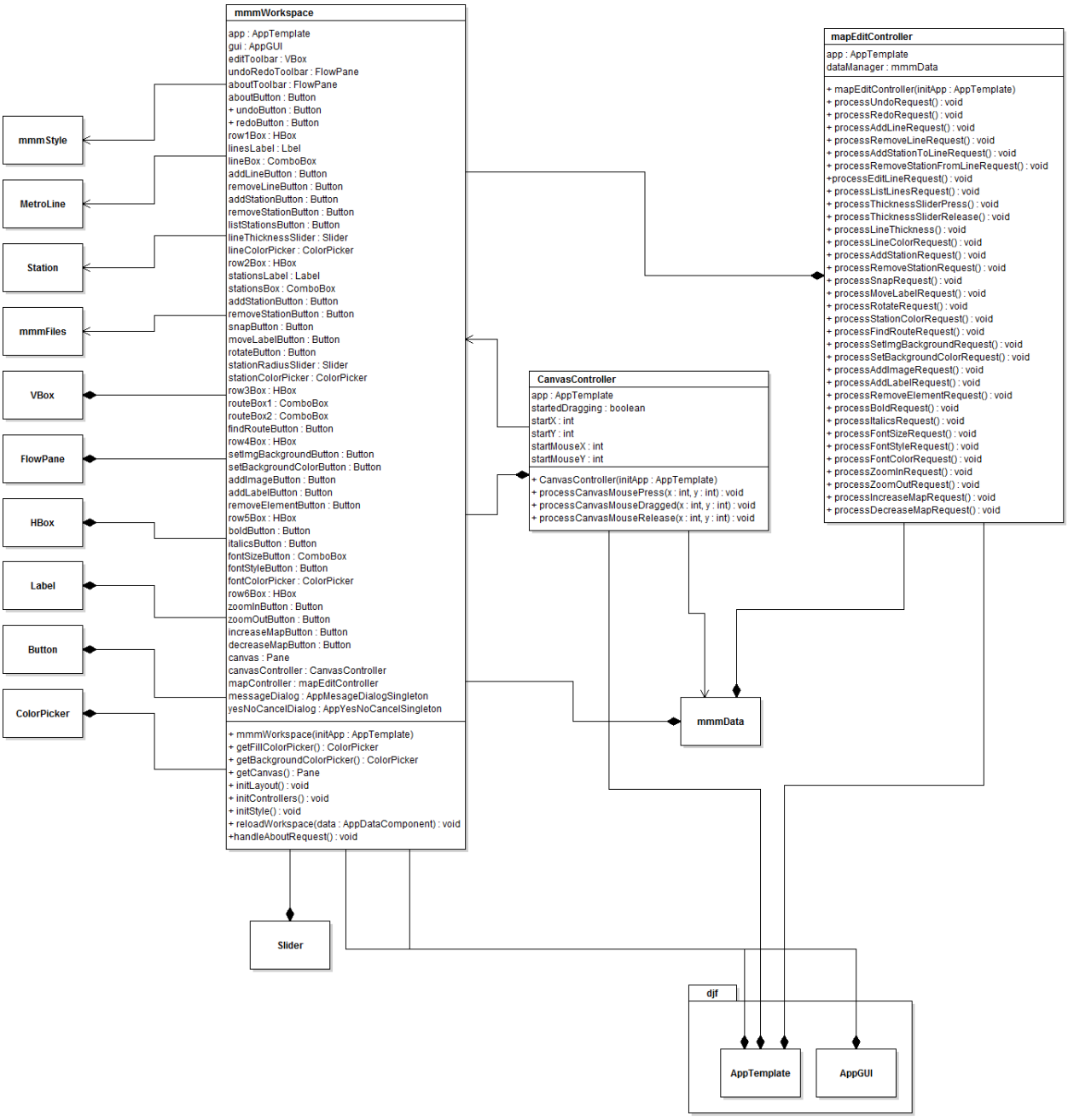


Figure 3.7: Detailed mmmWorkspace, CanvasController, and mapeEditController UML Class Diagrams

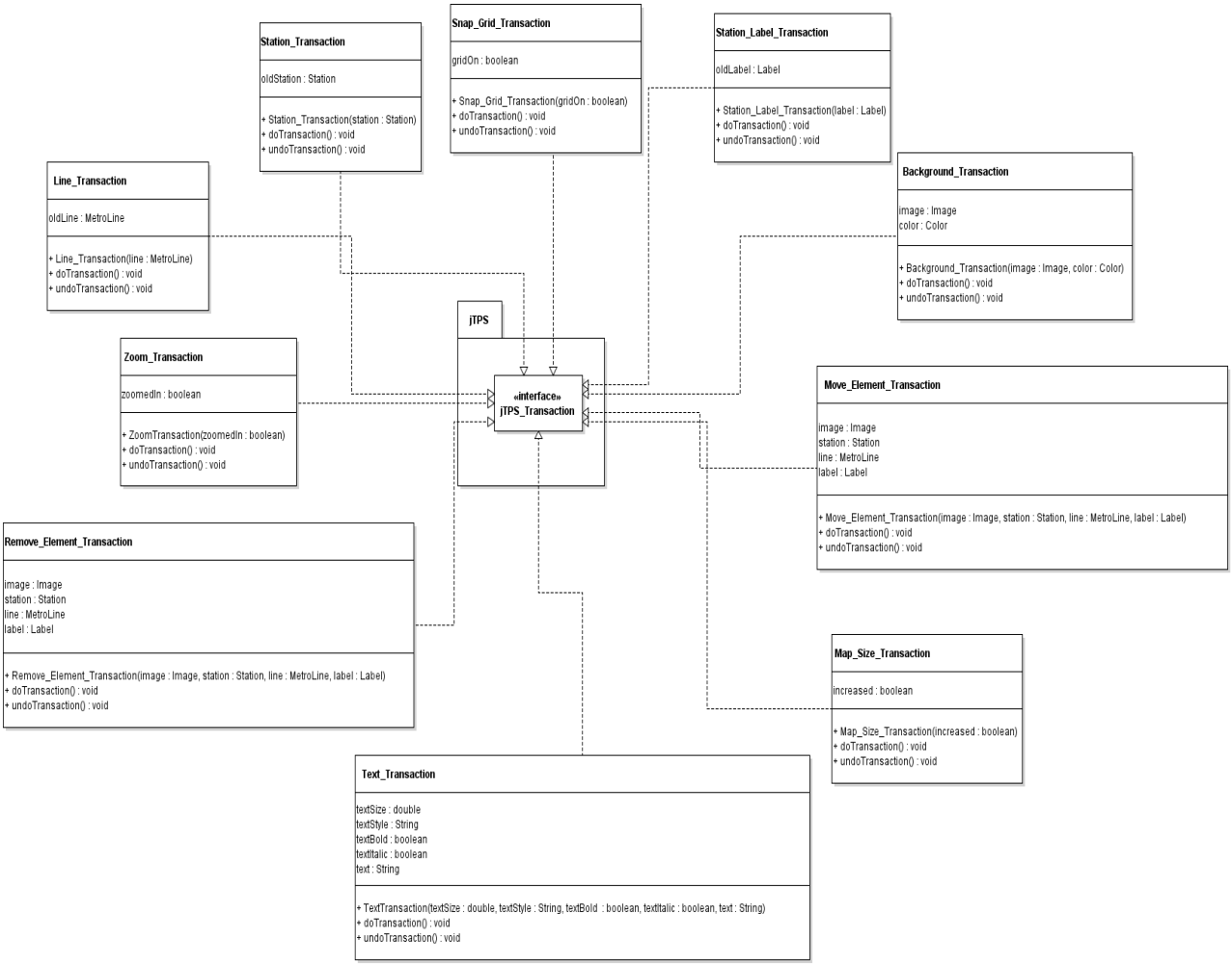
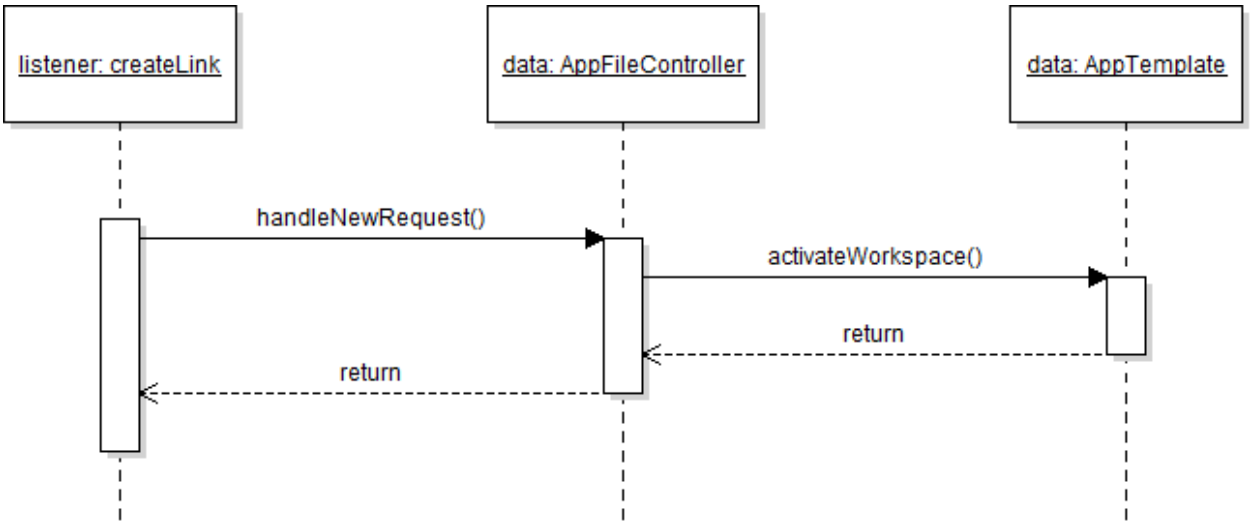


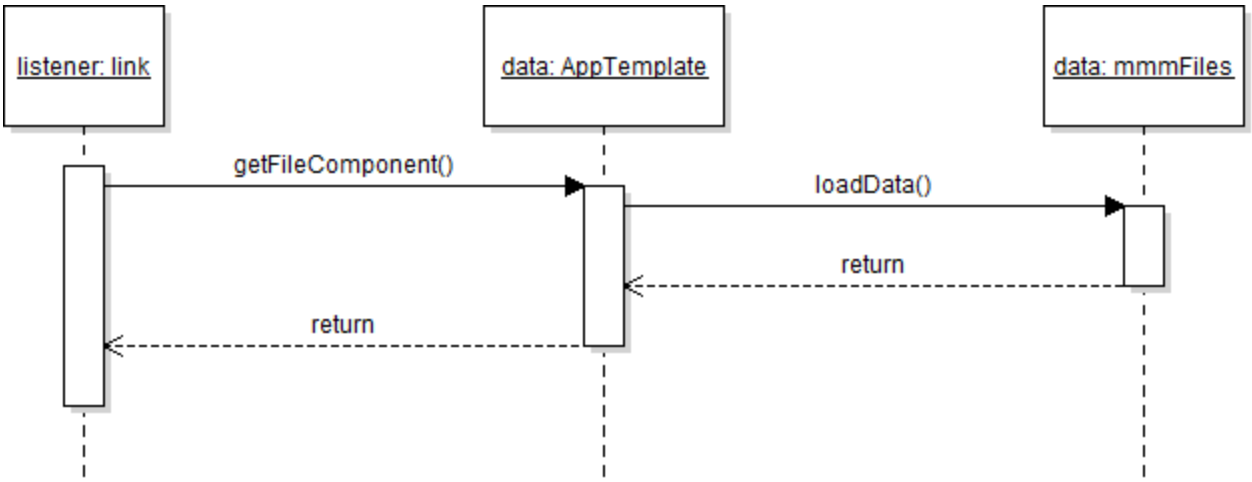
Figure 3.8: Detailed Transactions UML Class Diagram

4 Method-Level Design Viewpoint

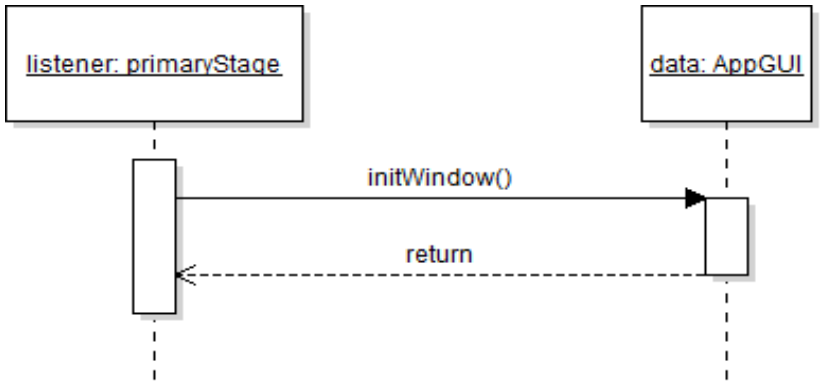
The following UML Sequence Diagrams demonstrate how methods within the program will interact with one another to fulfill certain use-cases.



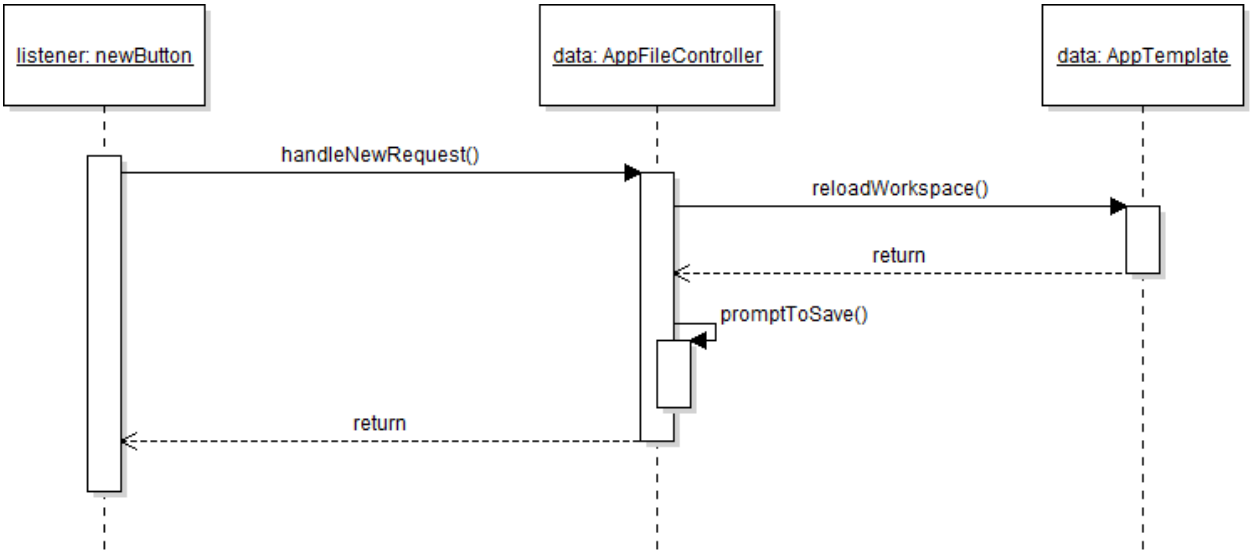
4.1: Create New Map (from welcome dialog) UML Sequence Diagram



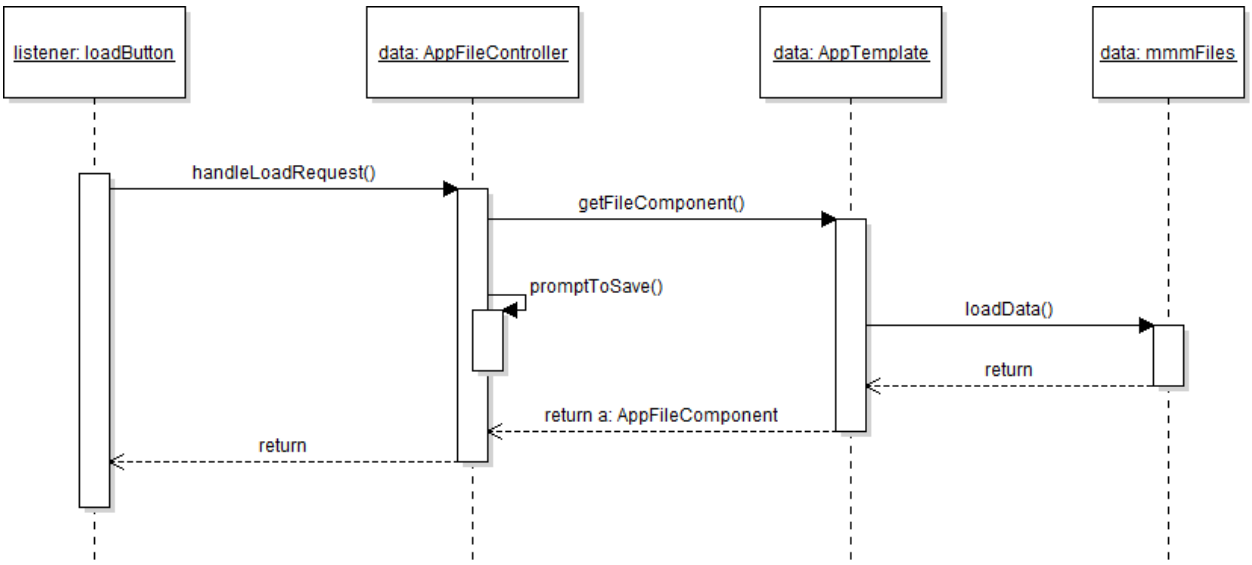
4.2: Select Recent Map to Load UML Sequence Diagram



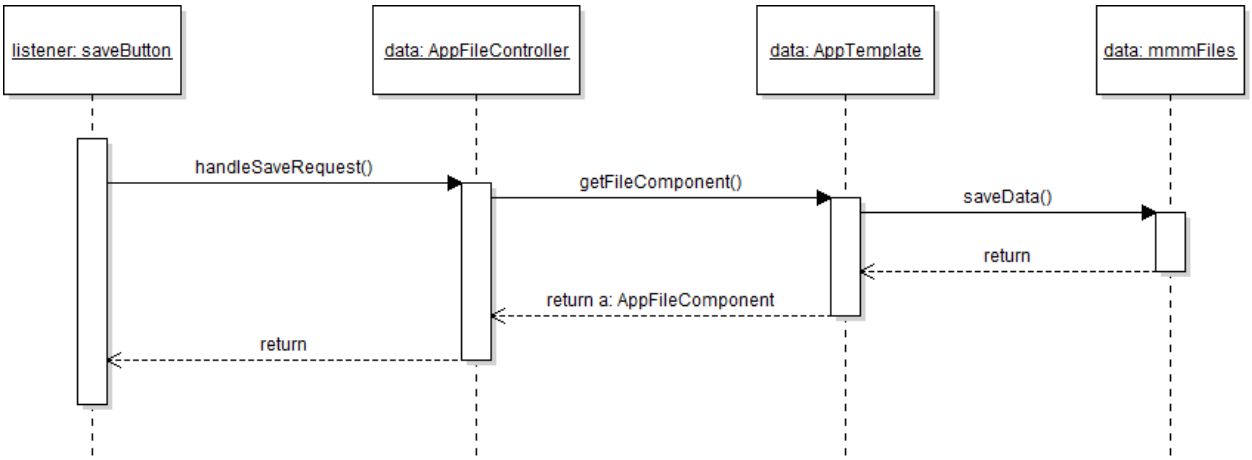
4.3: Close Welcome Dialog UML Sequence Diagram



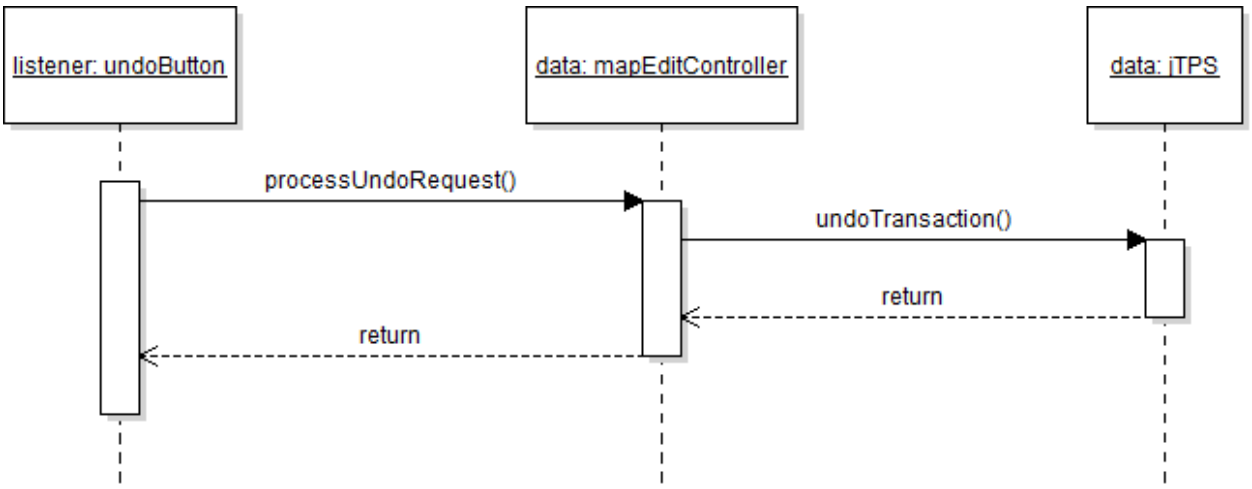
4.4: Create New Map (from main UI) Sequence Diagram



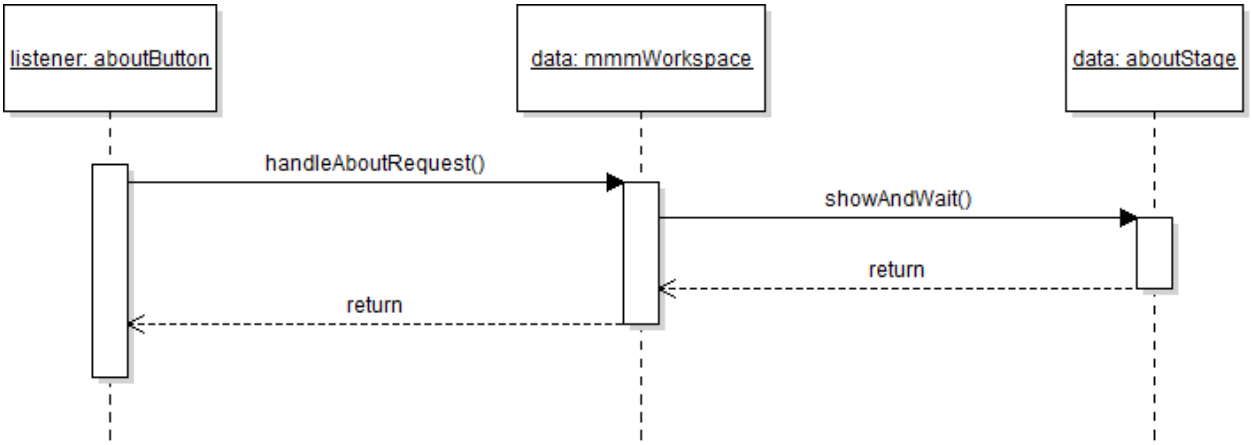
4.5: Load Map UML Sequence Diagram



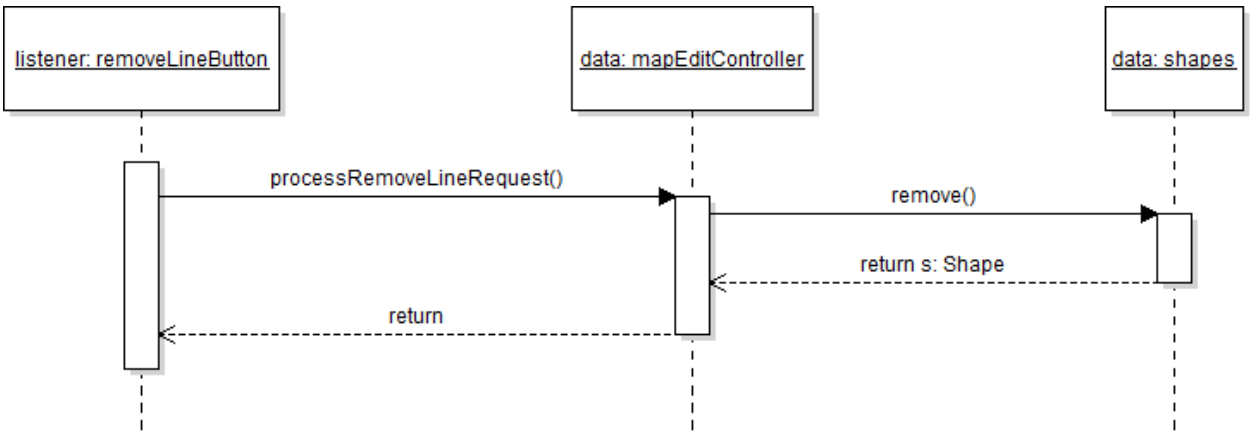
4.6: Save Map UML Sequence Diagram



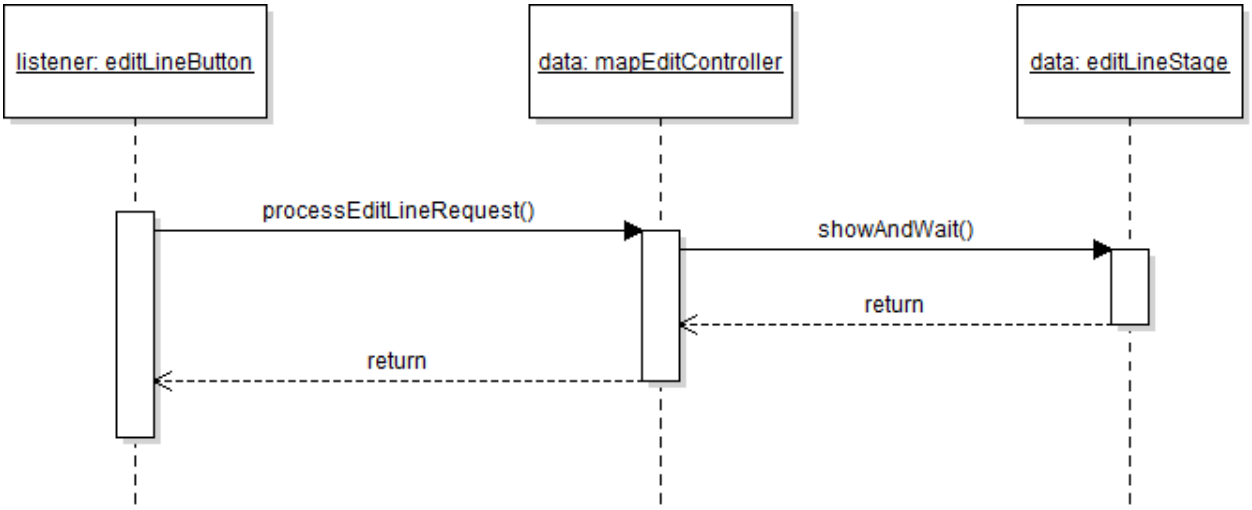
4.7: Undo Edit UML Sequence Diagram



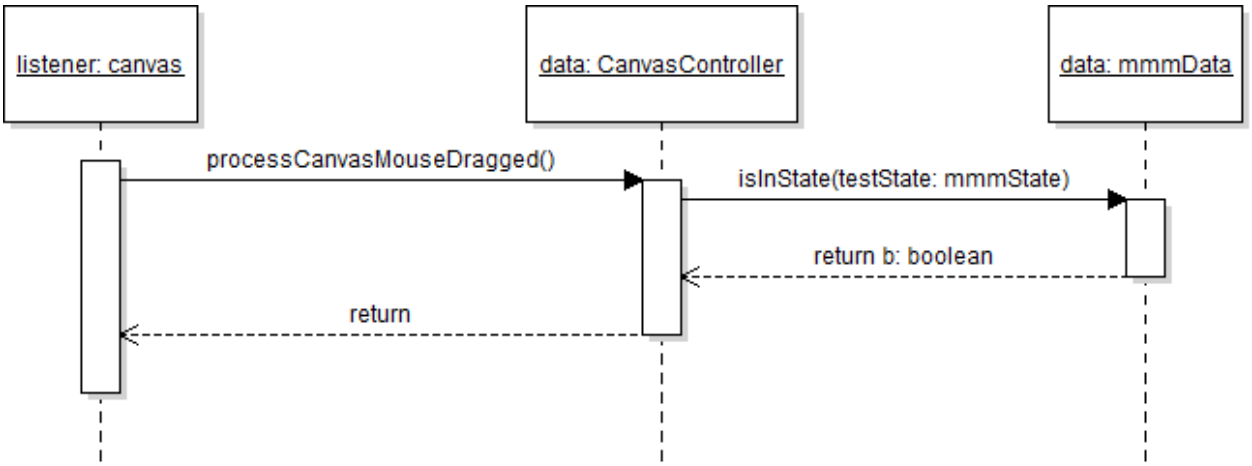
4.8: Learn About Application UML Sequence Diagram



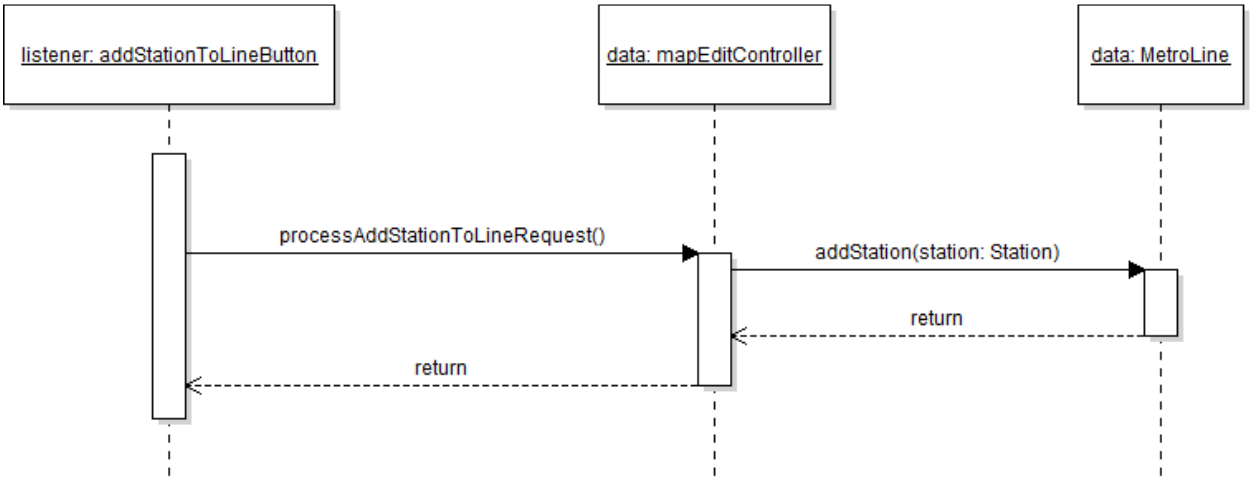
4.9: Remove Line UML Sequence Diagram



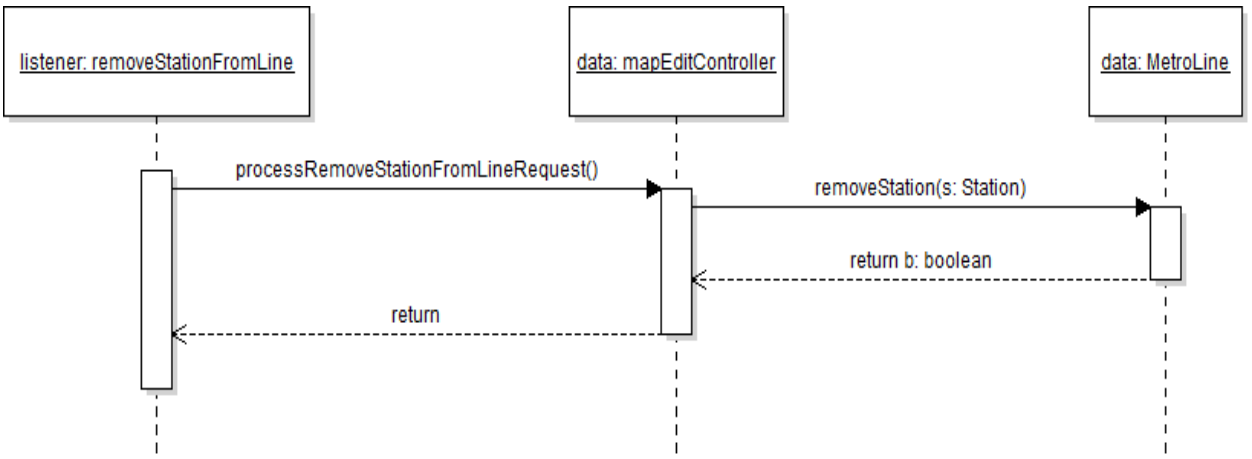
4.10: Edit Line UML Sequence Diagram



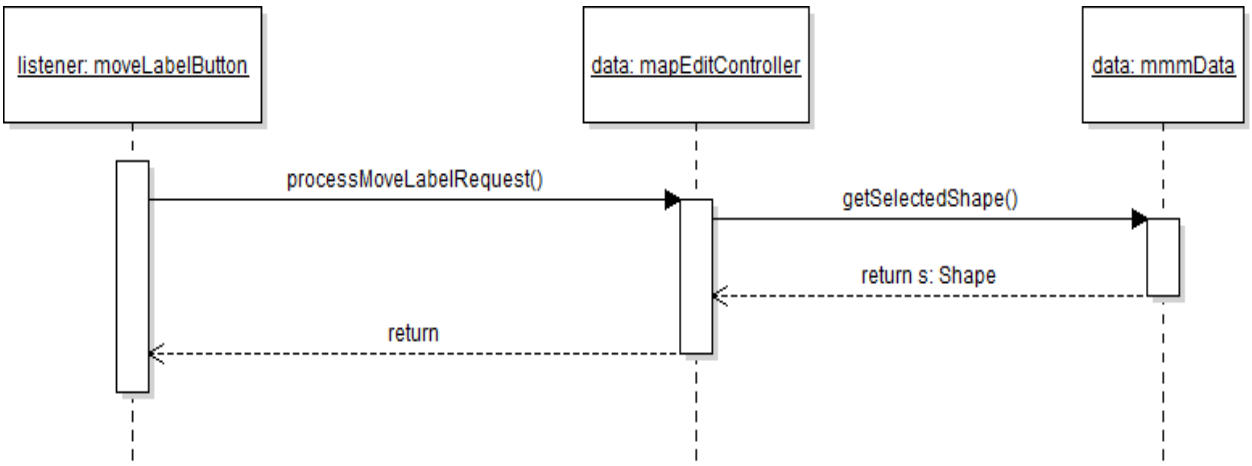
4.11: Move Line End UML Sequence Diagram



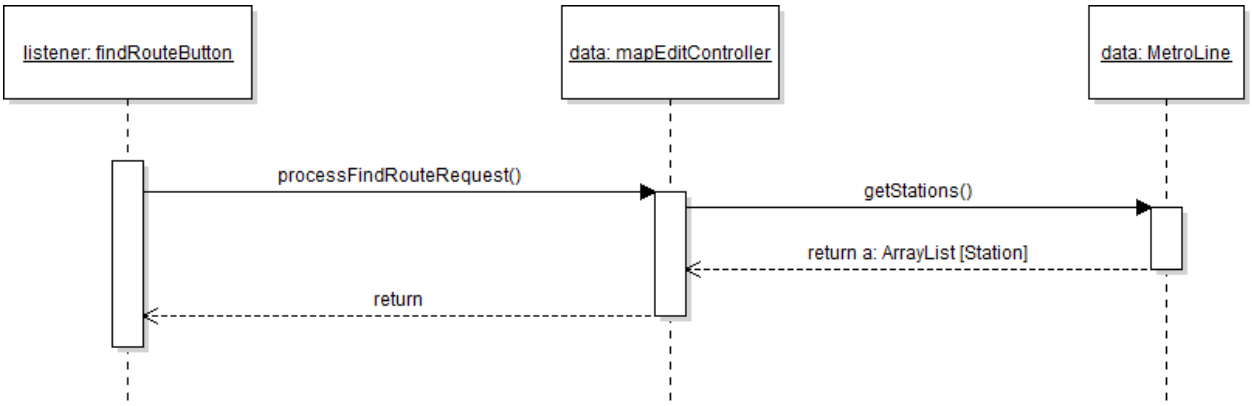
4.12: Add Stations to Line UML Sequence Diagram



4.13: Remove Stations from Line UML Sequence Diagram



4.14: Move Station Label UML Sequence Diagram



4.15: Find Route UML Sequence Diagram

5 File Structure and Formats

The three frameworks used, DesktopJavaFramework , PropertiesManager, and jTPS, will be provided inside jar files inside the MetroMapMaker directory. Images that are selected and imported into the application will be able to be accessed anywhere from the user's computer. Additionally, when a new file is created, the file will be saved within the application's work directory, and a new folder with the file's name will be created in the export directory. Any map that is exported from the application will have its image saved inside the corresponding folder inside the export directory, along with its corresponding JSON file. The structure for the application's export feature is as follows.

- When the export button is clicked, the application will convert the current state of the workspace into an image file, which will be placed inside the corresponding folder in the export directory.
- Then, the application will begin creating the JSON file using the JsonArrayBuilder class to create JSON objects for each piece of necessary data, i.e. shapes, locations, colors. etc.
- Finally, the array of objects will be stored inside a single JSON object, which will in turn be converted into a single file placed inside the export directory.

Figure 5.1: MetroMapMaker Export Description

As stated previously, the export feature will create a new image file and JSON file, placed inside the corresponding folder inside the export directory. The file structure below demonstrates this.

Export_directory

    map_directory

        image_file
        JSON_file

Figure 5.2: Export File Structure

6 Supporting Information

Below is a table of contents for navigating through the document.

6.1: Table of contents

6.2: Appendixes

N/A