**The Metro Map Maker™**

**Software Design Description**

**Author:** Hoi Yin To

Debugging Enterprises™

October, 2017

Version 1.0

**Abstract:** This document describes the software design for The Metro Map Maker, an application to build graphical representations of city subway systems

**Based on IEEE Std 1016™-2009 document format**

Copyright © 2017 Debugging Enterprises, which is a made up company and doesn’t really own The Metro Map Maker. Please note that this document is fictitious in that it simply serves as an example for CSE 219 students at Stony Brook University to use in developing their own SDD.

*No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher*

**1 Introduction**

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

* 1. **Purpose**

This document is to serve as the blueprint for the construction of the Metro Map Maker™ application. This design will use UML class diagrams to provide complete detail regarding all packages, classes, instance variables, class variables, and method signatures needed to build the application. In addition, UML Sequence diagrams will be used to specify object interactions post-initialization of the application, meaning in response to user interactions or timed events.

* 1. **Scope**

The goal of The Metro Map Maker™ is for users to easily make and edit subway maps. There will be an emphasis on ease of use. We will be using the provided framework DesktopJavaFramework and PropertiesManager to construct this application. There will be a common export format that will be provided for exported subway system data such that all maps can be used by a uniform application. Note that Java is the target language for this software design.

* 1. **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 an 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 a virtual machine layer between the Java application and the hardware to provide program portability.

**JavaScript** – the default scripting language of the Web, JavaScript is provided to pages in the form of text files with code that can be loaded and executed when a page loads so as to dynamically generate page content in the DOM.

**GUI** – Graphical User Interface, visual controls like buttons inside a window in a software application that collectively allow the user to operate the program.

**Stylesheet** – a static text file employed by HTML pages that can control the colors, fonts, layout and other style components in a Web page.

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

**Use Case Descriptions** – A formal format for specifying how a user will interact with a system.

**Sequence Diagram** – A UML document format that specifies how object methods interact with one another. Sprite – a renderable, and sometimes movable or clickable image in the game. Each Sun, Zombie, and Brain will be its own Sprite, as will GUI controls.

**Map** - A diagrammatic representation of an area of land or sea showing physical features, cities, roads, etc.

**Metro Map** - A topological map in the form of a schematic diagram used to illustrate the routes and stations within a public transport system

**Metro Map Maker** - An application which provides the user with a set of tools to build graphical representations of city subway systems with named lines and named stops and intersecting lines and landmarks. It will also provide a means for calculating the best route to take to journey from one particular station to another.

* 1. **References**

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

**The Metro Map Maker™** **SRS** – Debugging Enterprises’ Software Requirements Specification for the Metro Map Maker application.

**1.5 Overview**

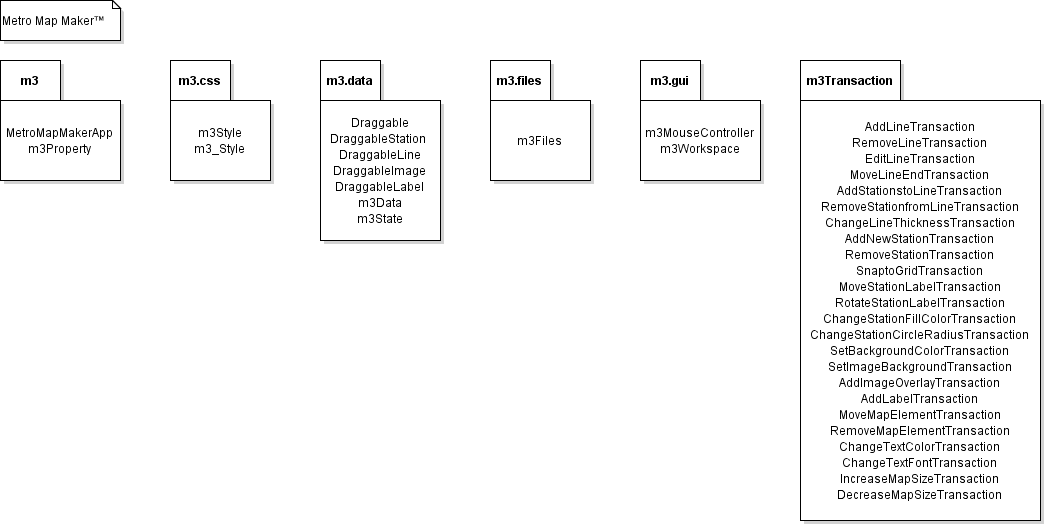
This Software Design Description document provides a working design for the Metro Map Maker™ between components before proceeding with the implementation stage. Section 2 of this document will provide the Package-Level Viewpoint, specifying the packages and frameworks to be designed. Section 3 will provide the Class-Level Viewpoint, using UML Class Diagrams to specify how the classes should be constructed. Section 4 will provide the Method-Level System Viewpoint, describing how methods will interact with one another. Section 5 provides deployment information like file structures and formats to use. Section 6 provides a Table of Contents, an Index, and References. Note that all UML Diagrams in this document were created using the VioletUML editor.

1. **Package-Level Design Viewpoint**

As mentioned, this design will encompass the Metro Map Maker™ application and the provided framework DesktopJavaFramework and PropertiesManager to be used in its construction. In building the application we will heavily rely on the Java API to provide services. Following are descriptions of the components to be built, as well as how the Java API will be used to build them.

* 1. **Metro Map Maker™ overview**

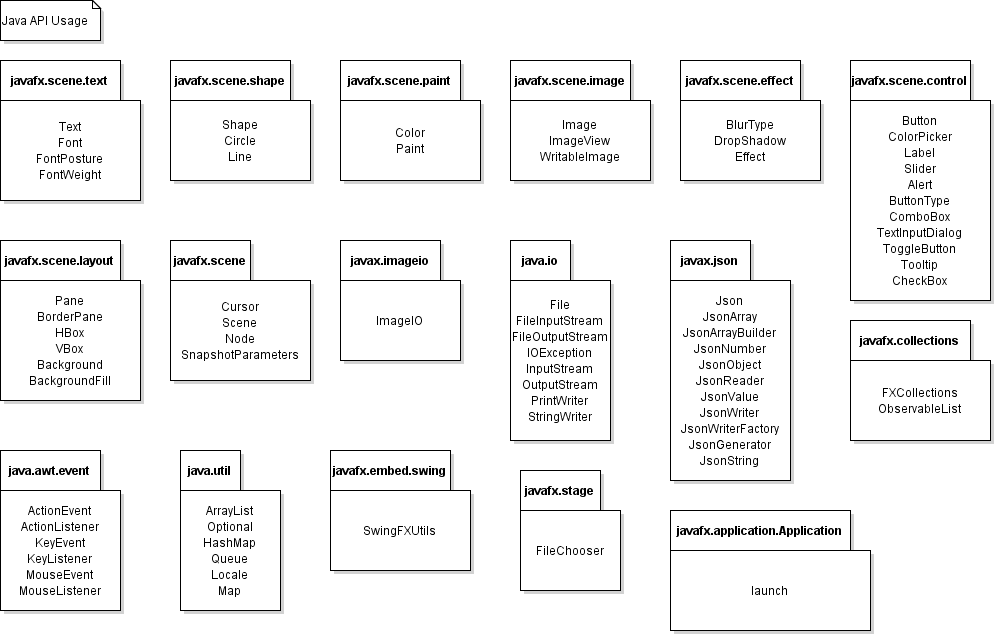
Figure 2.1 specifies all the components to be developed and places all classes in home packages.



**Figure 2.1: Design Packages Overview**

* 1. **Java API Usage**

Both the framework and the application will be developed using the Java programming languages. As such, this design will make use of the classes specified in Figure 2.2.



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

* 1. **Java API Usage Descriptions**

Tables 2.1-2.17 below summarize how each of these classes will be used.

|  |  |
| --- | --- |
| Class/Interface | Use |
| Text | For displaying text |
| Font | For setting the fonts for rendered text |
| FontPosture | For setting the font posture for rendered text |
| FontWeight | For setting the font weight for rendered text |

**Table 2.1: Uses for classes in the Java API’s javafx.scene.text package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| Shape | For creating shapes |
| Circle | For creating circles |
| Line | For creating lines |

**Table 2.2: Uses for classes in the Java API’s javafx.scene.shape package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| Color | For setting the rendering colors for text, shapes, background and lines. |
| Paint | For setting the rendering colors for text, shapes, background and lines. |

**Table 2.3: Uses for classes in the Java API’s javafx.scene.paint package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| Image | For storing image data |
| ImageView | For displaying image |
| WritableImage | For writing pixels directly to an image. |

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

|  |  |
| --- | --- |
| Class/Interface | Use |
| Effect | For enhancing the appearance on nodes visually. |
| BlurType | For blurring the shadow |
| DropShadow | For rendering a shadow of the content to which it is applied |

**Table 2.5: Uses for classes in the Java API’s javafx.scene.effect package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| Button | For processing an action when a user clicks on it |
| ColorPicker | For selecting a particular color from the available range |
| Label | For displaying a text element |
| Slider | For adjusting the line thickness |
| Alert | For showing dialogs |
| ButtonType | For specifying which buttons should be shown to users in the dialogs |
| ComboBox | For enabling the user to select an option |
| TextInputDialog | For entering a text content by users |
| ToggleButton | For choosing options by users |
| Tooltip | For showing additional information about a Node |
| CheckBox | For allowing users to make choices |

**Table 2.6: Uses for classes in the Java API’s javafx.scene.control package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| Pane | For containing different interface components |
| BorderPane | For containing different interface components |
| HBox | For containing different interface components |
| VBox | For containing different interface components |
| Background | For setting the background color |
| BackgroundFill | For setting the background color |

**Table 2.7: Uses for classes in the Java API’s javafx.scene.layout package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| Cursor | For encapsulating the bitmap representation of the mouse cursor |
| Scene | For  containing all content in a scene graph |
| Node | For serving as base class for scene graph nodes |
| SnapshotParameters | For snapshots |

**Table 2.8: Uses for classes in the Java API’s javafx.scene package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| ImageIO | For snapshots |

**Table 2.9: Uses for classes in the Java API’s javax.imageio package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| File | For storing a file |
| FileInputStream | For loading a file |
| FileOutputStream | For saving a file |
| IOException | For throwing exception |
| InputStream | For loading a file |
| OutputStream | For saving a file |
| PrintWriter | For saving a file |
| StringWriter | For saving a file |

**Table 2.10: Uses for classes in the Java API’s java.io package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| Json | For saving data |
| JsonArray | For saving data |
| JsonArrayBuilder | For saving data |
| JsonNumber | For saving data |
| JsonObject | For saving data |
| JsonReader | For saving data |
| JsonValue | For saving data |
| JsonWriter | For saving data |
| JsonWriterFactory | For saving data |
| JsonGenerator | For saving data |
| JsonString | For saving data |

**Table 2.11: Uses for classes in the Java API’s javax.json package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| FXCollections | For storing data of combo box |
| ObservableList | For storing the nodes |

**Table 2.12: Uses for classes in the Java API’s javafx.collectionspackage**

|  |  |
| --- | --- |
| Class/Interface | Use |
| ActionEvent | For getting information about an action event like which button was pressed. |
| ActionListener | For responding to an action event, like a button press. We will provide our own custom implementation of this interface. |
| KeyEvent | For getting information about a key event, like which key was pressed. |
| KeyListener | For responding to a key event, like a key press. We will provide our own custom implementation of this interface |
| MouseEvent | For getting information about a mouse event, like where was the mouse pressed? |
| MouseListener | For responding to a mouse event, like a mouse button press. We will provide our own custom implementation of this interface. |

**Table 2.13: Uses for classes in the Java API’s java.awt.event package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| ArrayList | For storing the stations |
| Optional | For containing a non-null reference to another object |
| HashMap | For storing (name,value) key pairs, we’ll use it for storing our stations, accessible using their names. |
| Queue | For storing data |
| Locale | For executing the program |
| Map | For storing (name,value) key pairs, we’ll use it for storing our stations, accessible using their names. |

**Table 2.14: Uses for classes in the Java API’s java.util package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| SwingFXUtils | For snapshots |

**Table 2.15: Uses for classes in the Java API’s javafx.embed.swing package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| FileChooser | For choosing a file |

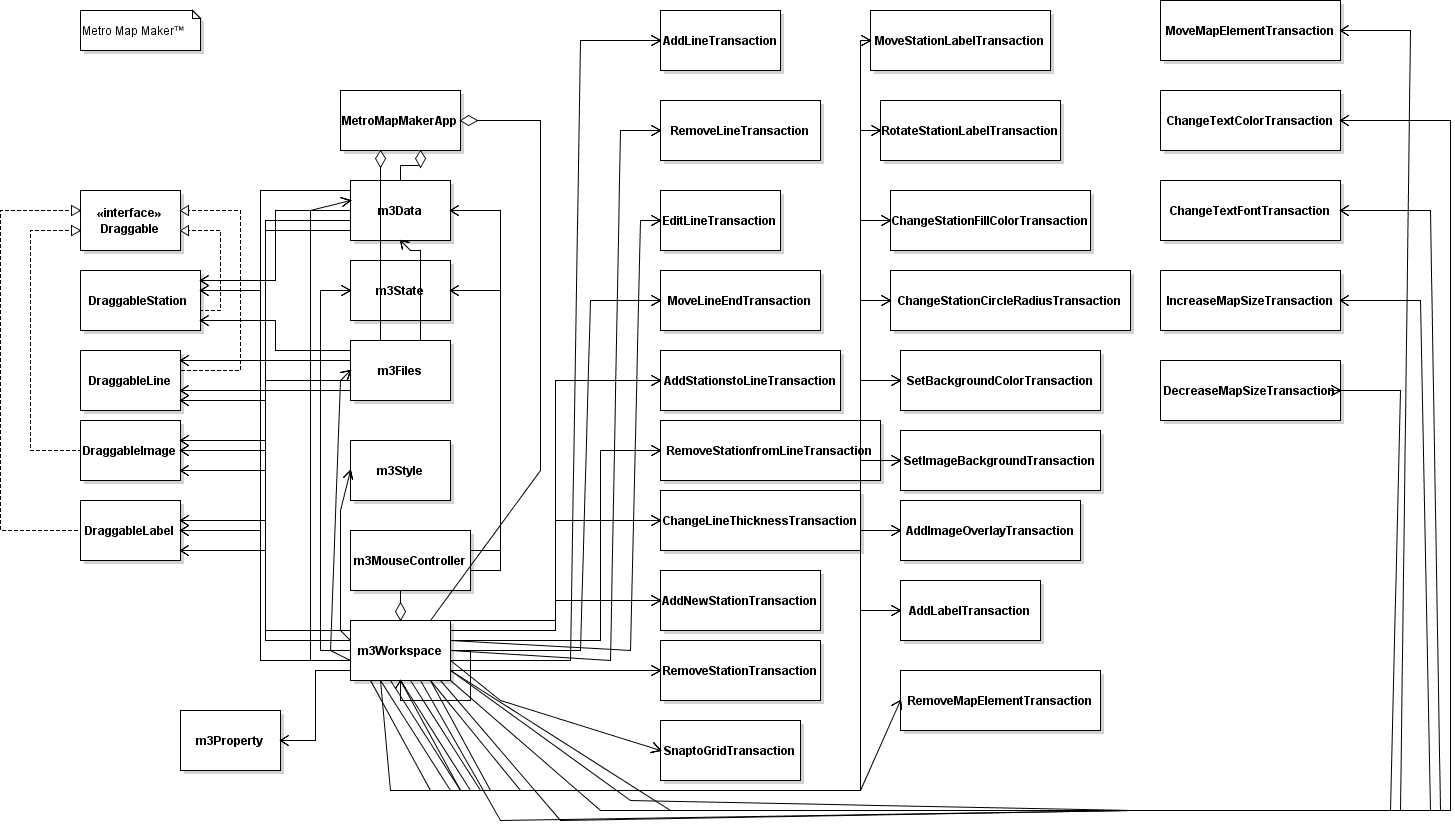
**Table 2.16: Uses for classes in the Java API’s javafx.stage package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| launch | For opening the JavaFX |

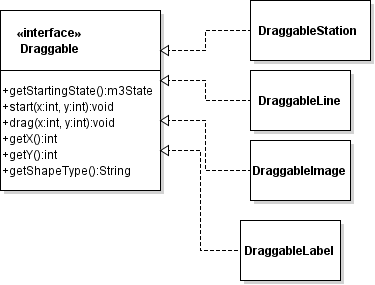
**Table 2.17: Uses for classes in the Java API’s javafx.application.Application package**

**3 Class-Level Design Viewpoint**

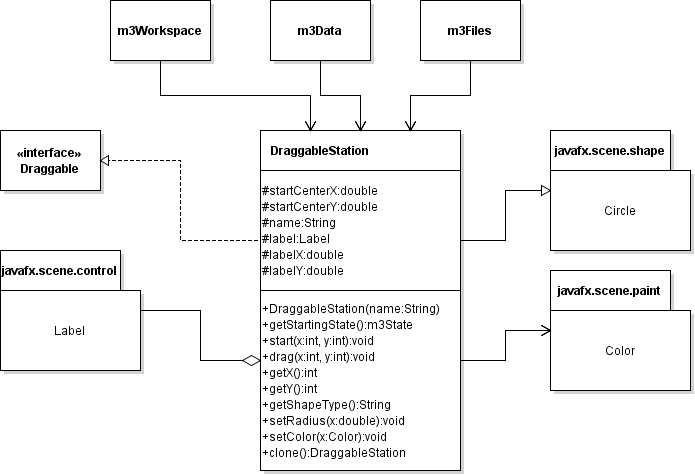
The following UML Class Diagrams reflect the design of the Metro Map Maker™ application. Note that due to the complexity of the project, we present the class designs using a series of diagrams going from overview diagrams down to detailed ones.

****

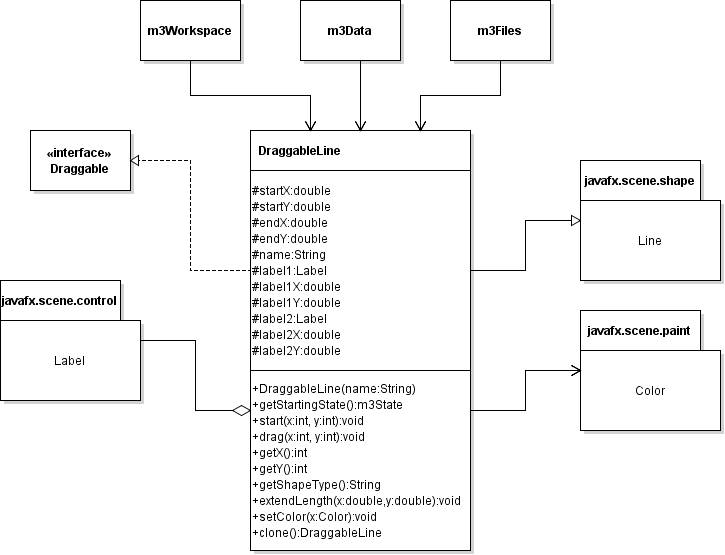
**Figure 3.1: Metro Map Maker™** **Overview UML Class Diagram**

****

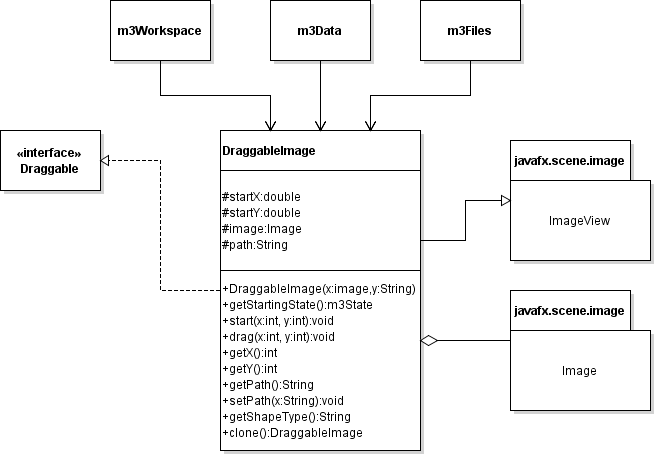
**Figure 3.2: Detailed Draggable UML Class Diagram**

****

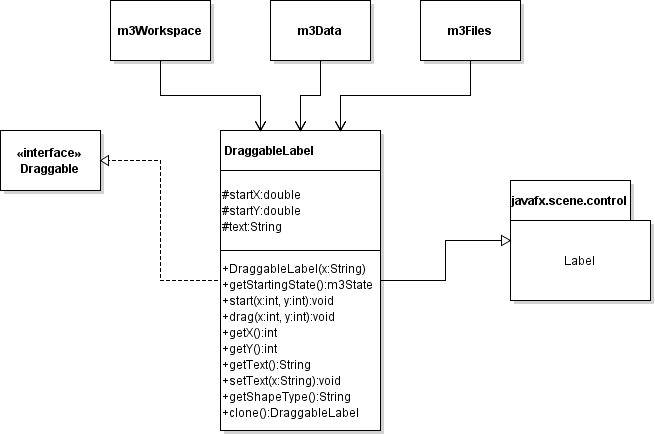
**Figure 3.3: Detailed DraggableStation UML Class Diagram**

****

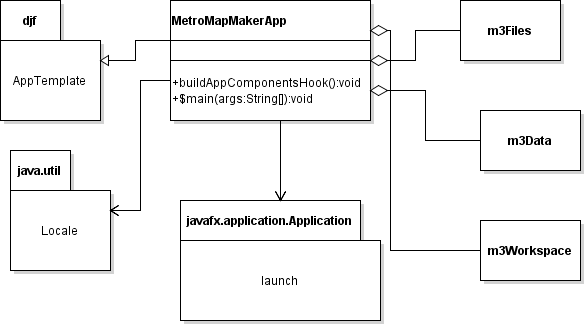
**Figure 3.4: Detailed DraggableLine UML Class Diagram**

****

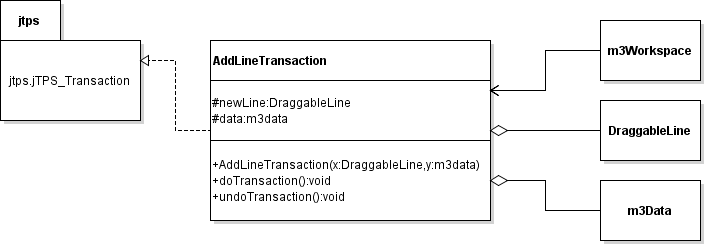
**Figure 3.5: Detailed DraggableImage UML Class Diagram**

****

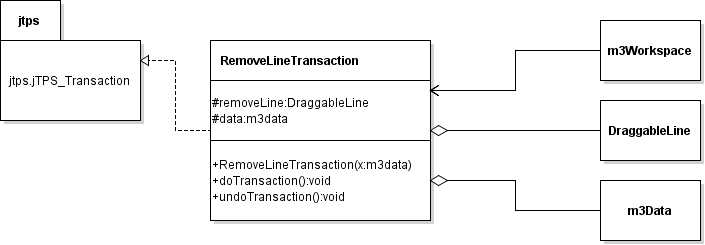
**Figure 3.5: Detailed DraggableLabel UML Class Diagram**

****

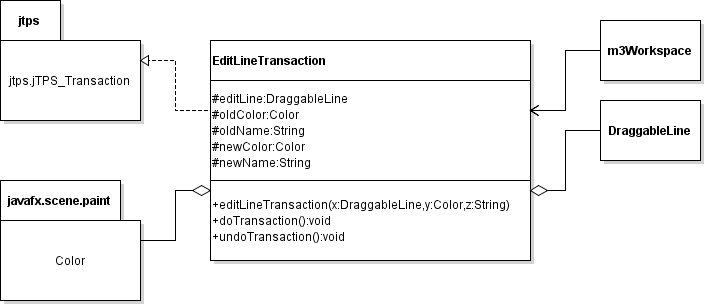
**Figure 3.6: Detailed MetroMakerApp UML Class Diagram**

****

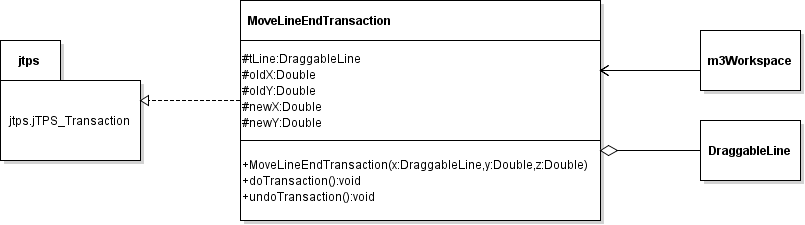
**Figure 3.7: Detailed AddLineTransaction UML Class Diagram**

****

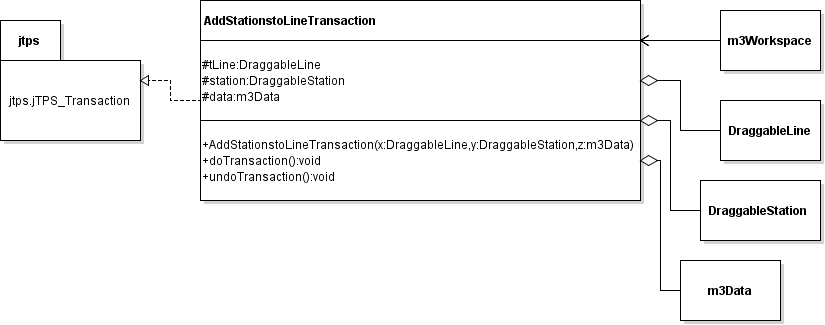
**Figure 3.8: Detailed RemoveLineTransaction UML Class Diagram**

****

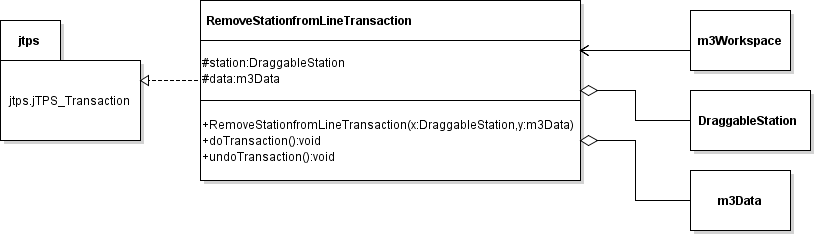
**Figure 3.9: Detailed EditLineTransaction UML Class Diagram**

****

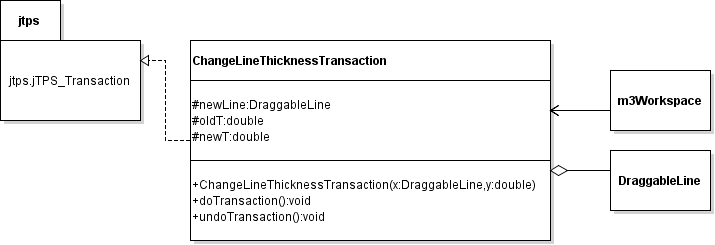
**Figure 3.10: Detailed MoveLineEndTransaction UML Class Diagram**

****

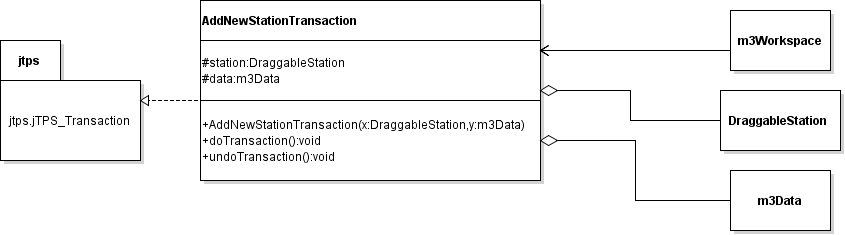
**Figure 3.11: Detailed AddStationstoLineTransaction UML Class Diagram**

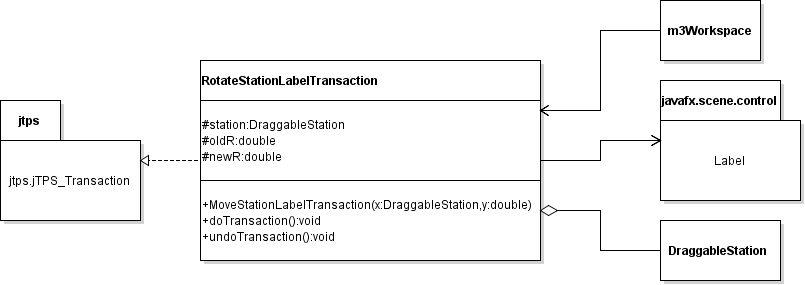
****

**Figure 3.12: Detailed RemoveStationfromLineTransaction UML Class Diagram**

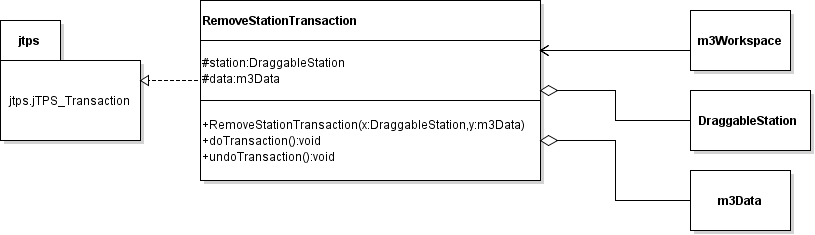
****

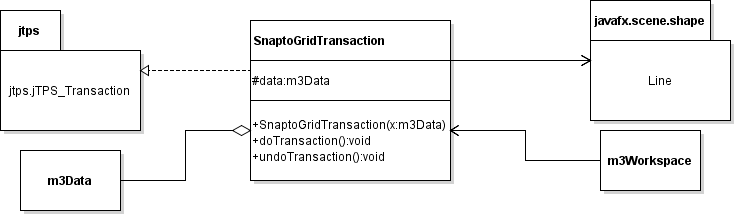
**Figure 3.13: Detailed ChangeLineThicknessTransaction UML Class Diagram**

****

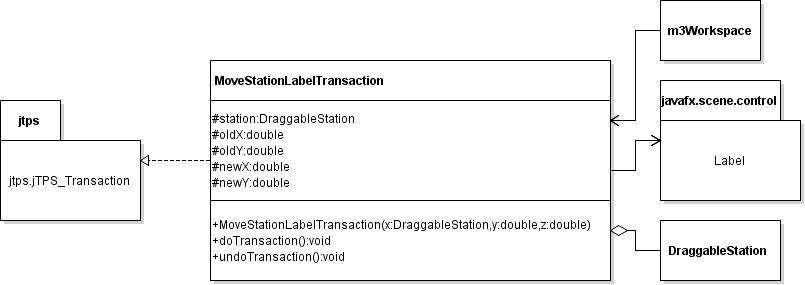
**Figure 3.14: Detailed AddNewStationTransaction UML Class Diagram**

**Figure 3.15: Detailed RotateStationLabelTransaction UML Class Diagram**

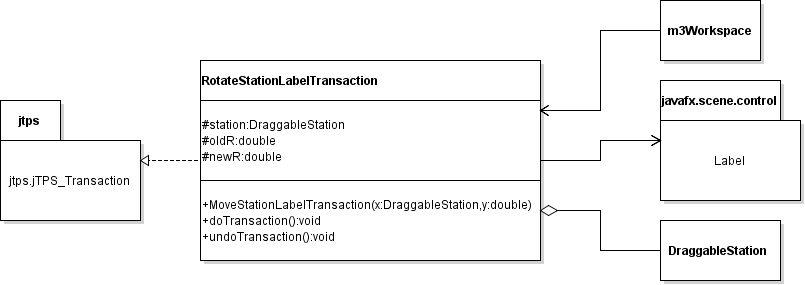
** Figure 3.16: Detailed RemoveStationTransaction UML Class Diagram**

****

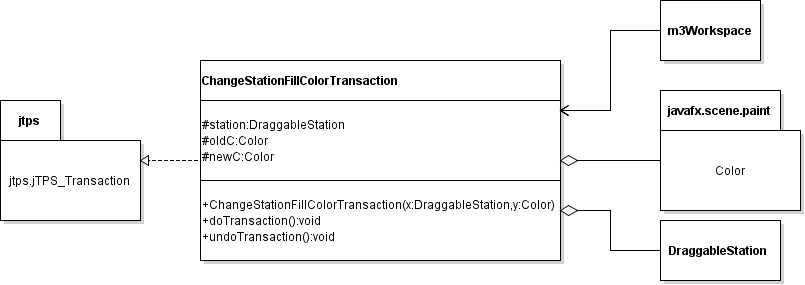
**Figure 3.17: Detailed SnaptoGridTransaction UML Class Diagram**

****

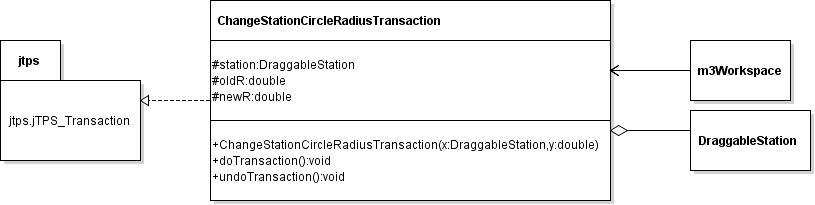
**Figure 3.18: Detailed MoveStationLabelTransaction UML Class Diagram**

****

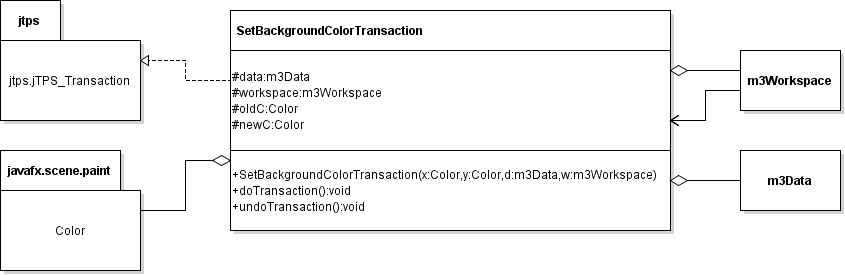
**Figure 3.19: Detailed RotateStationLabelTransaction UML Class Diagram**

****

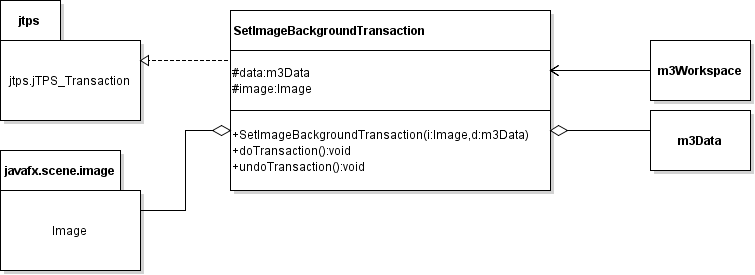
**Figure 3.20: Detailed ChangeStationFillColorTransaction UML Class Diagram**

****

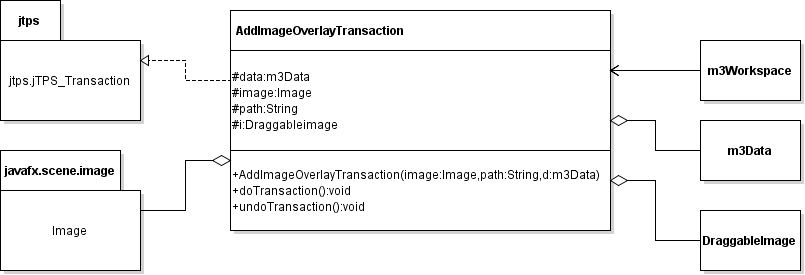
**Figure 3.21: Detailed ChangeStationCircleRadiusTransaction UML Class Diagram**

****

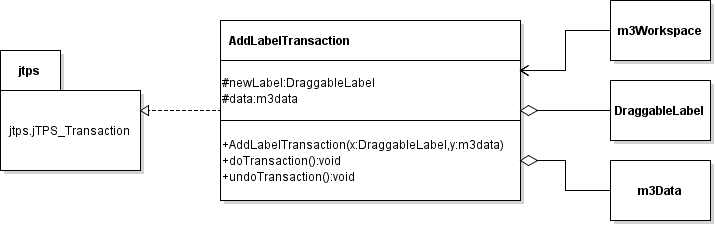
**Figure 3.22: Detailed SetBackgroundColorTransaction UML Class Diagram**

****

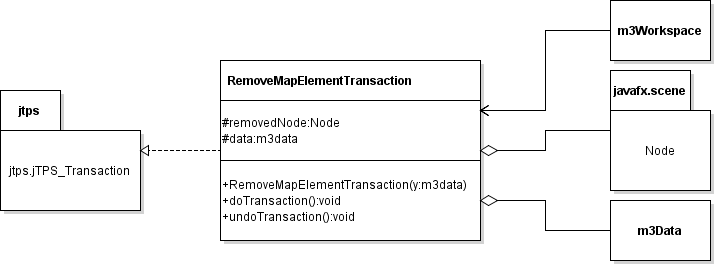
**Figure 3.23: Detailed SetImageBackgroundTransaction UML Class Diagram**

****

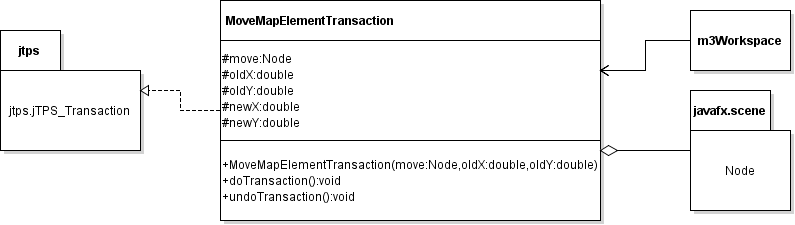
**Figure 3.24: Detailed AddImageOverlayTransaction UML Class Diagram**

****

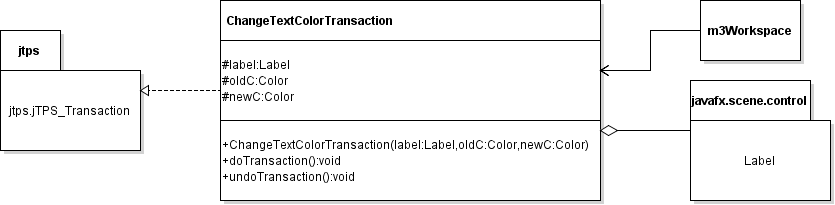
**Figure 3.25: Detailed AddLabelTransaction UML Class Diagram**

****

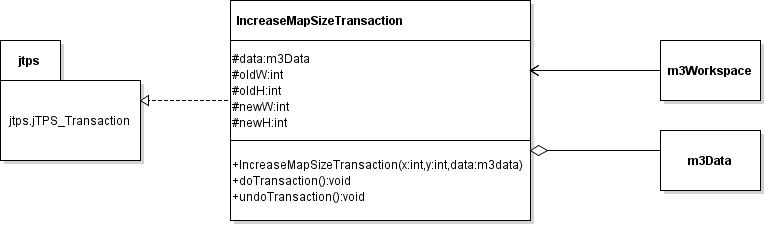
**Figure 3.26: Detailed RemoveMapElementTransaction UML Class Diagram**

****

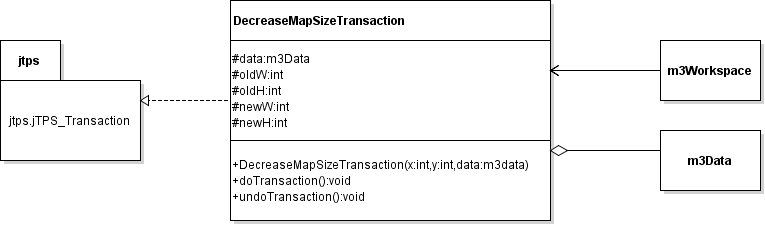
**Figure 3.27: Detailed MoveMapElementTransaction UML Class Diagram**

****

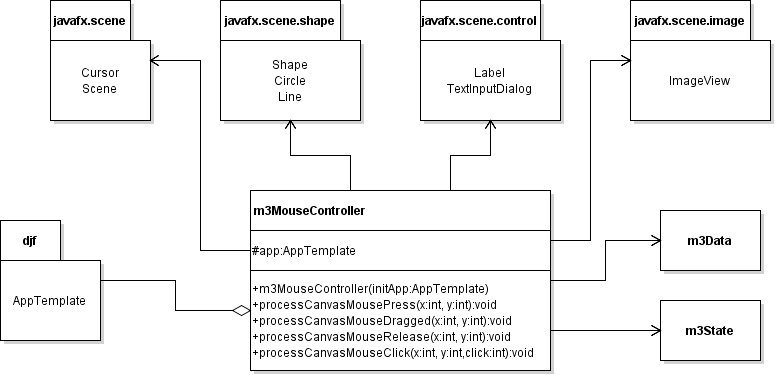
**Figure 3.28: Detailed ChangeTextColorTransaction UML Class Diagram**

****

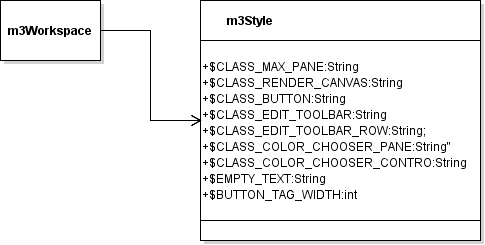
**Figure 3.29: Detailed IncreaseMapSizeTransaction UML Class Diagram**

****

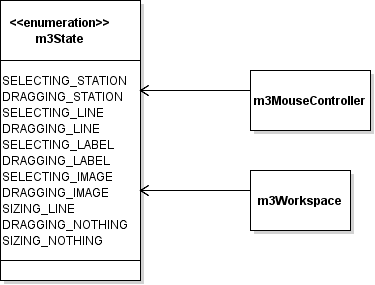
**Figure 3.30: Detailed DecreaseMapSizeTransaction UML Class Diagram**

****

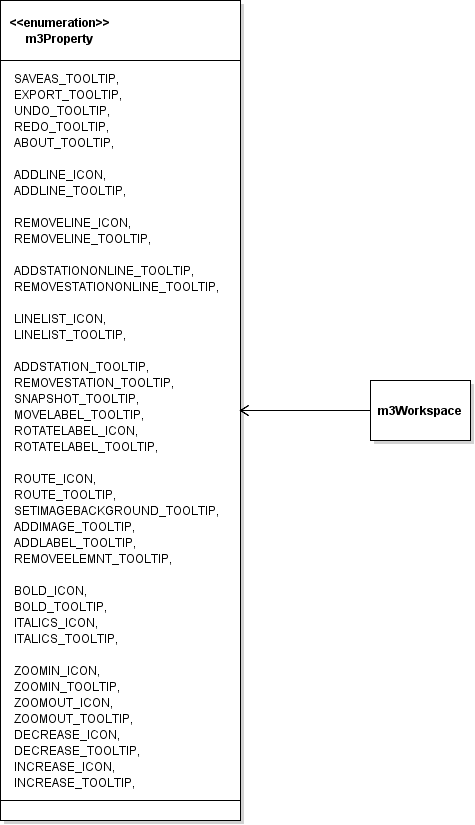
**Figure 3.31: Detailed m3MouseController UML Class Diagram**

****

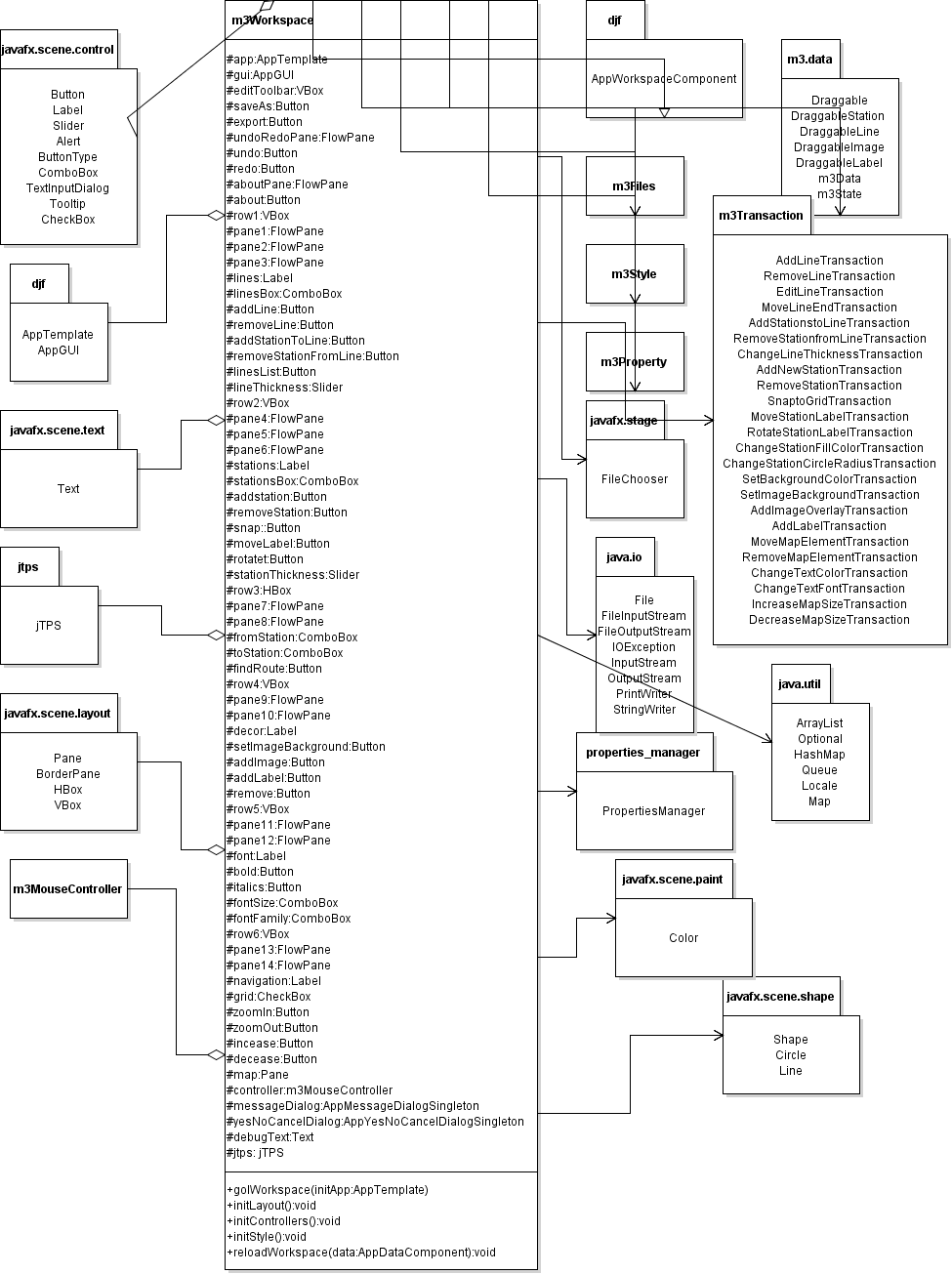
**Figure 3.32: Detailed m3Style UML Class Diagram**

****

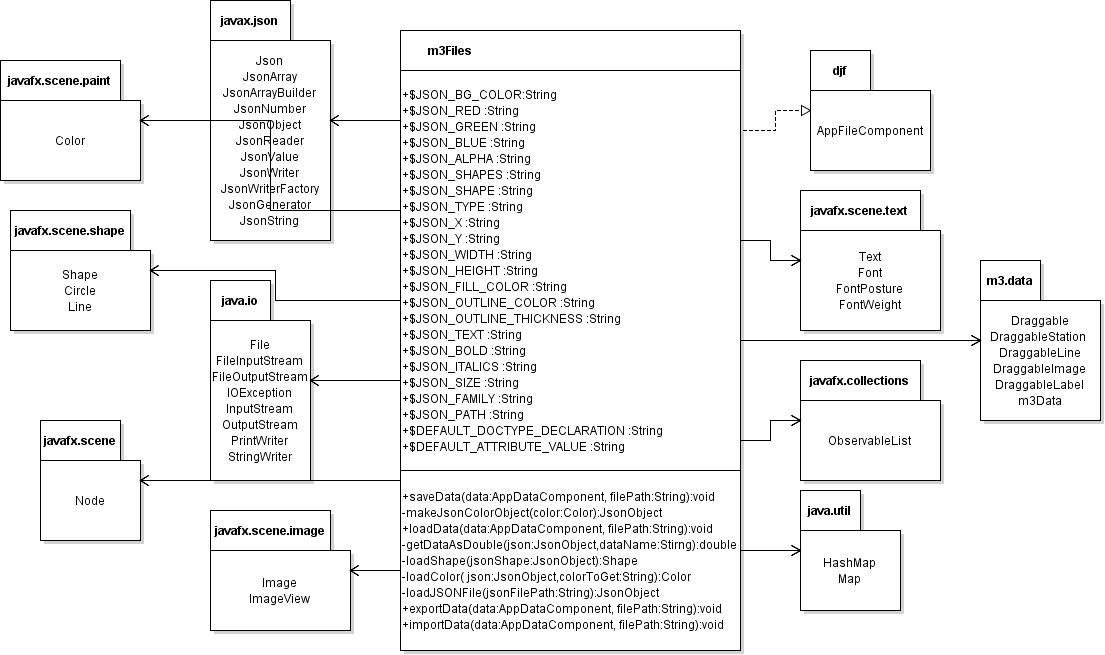
**Figure 3.33: Detailed m3State UML Class Diagram**

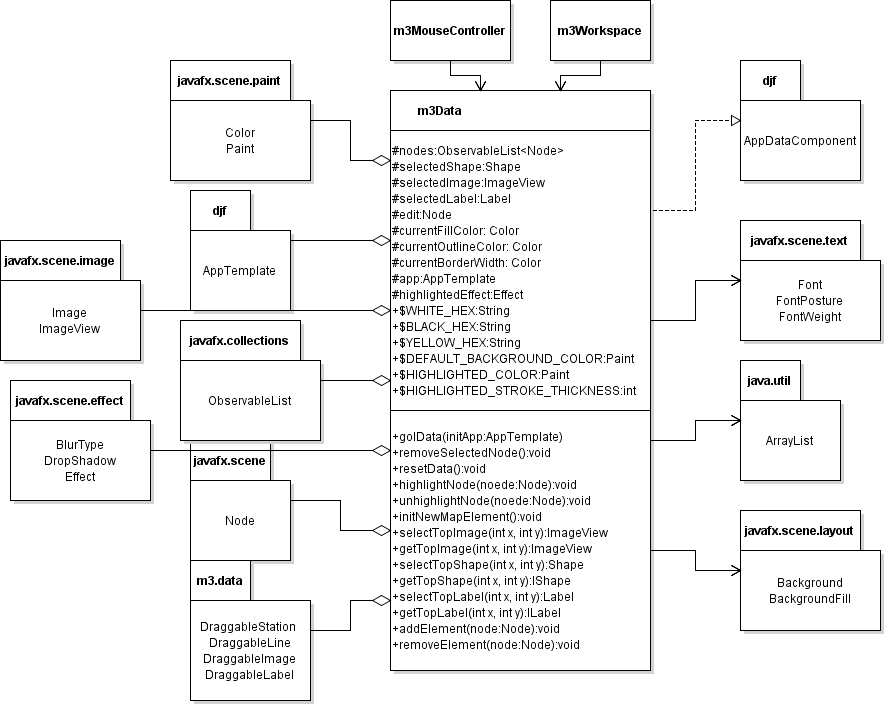
****

**Figure 3.34: Detailed m3Property UML Class Diagram**

****

**Figure 3.35: Detailed m3Workspace UML Class Diagram**

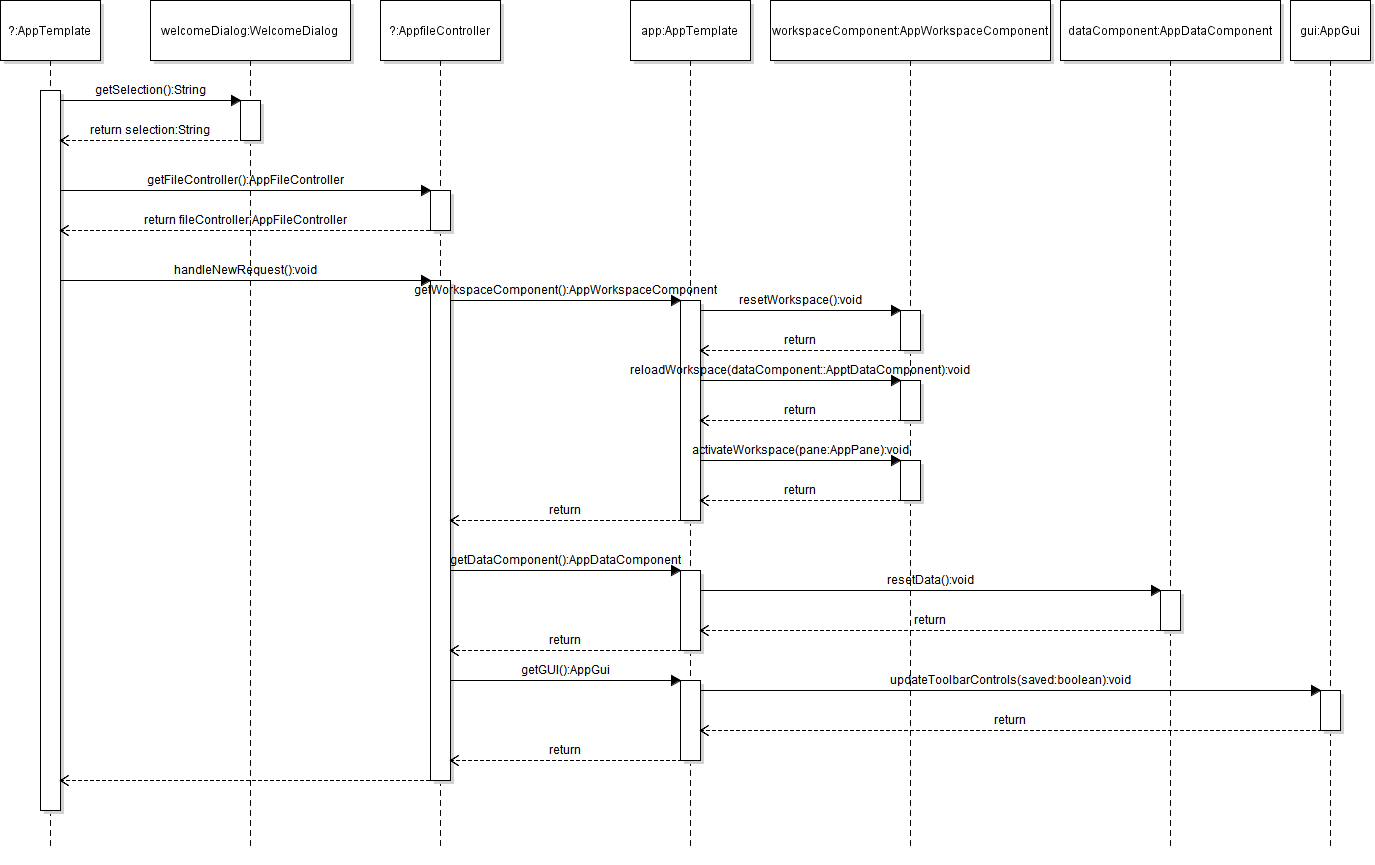
**Figure 3.36: Detailed m3Workspace UML Class Diagram**

****

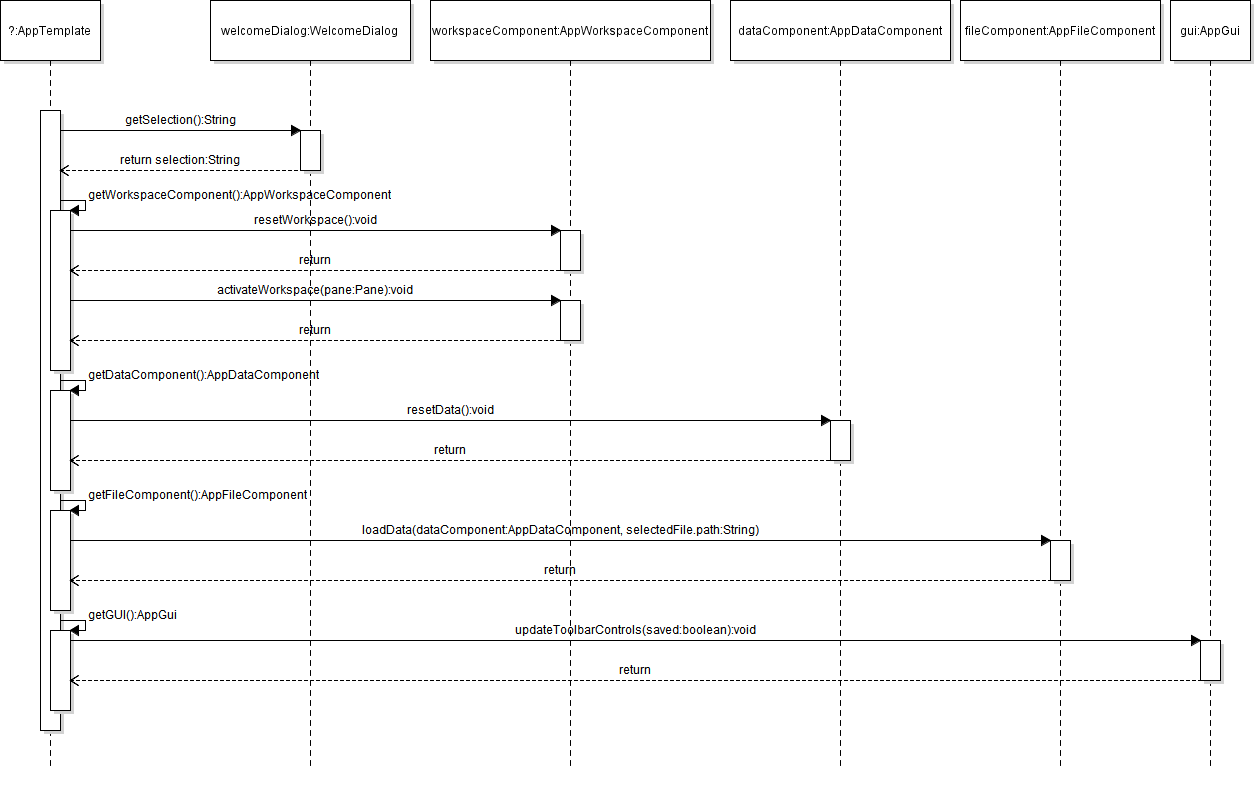
**Figure 3.37: Detailed m3Data UML Class Diagram**

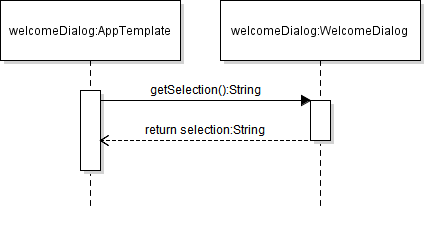
1. **Method-Level Design Viewpoint**

Now that the general architecture of the classes has been determined, it is time to specify how data will flow through the system. The following UML Sequence Diagrams describe the methods called within the code to be developed in order to provide the appropriate event responses.

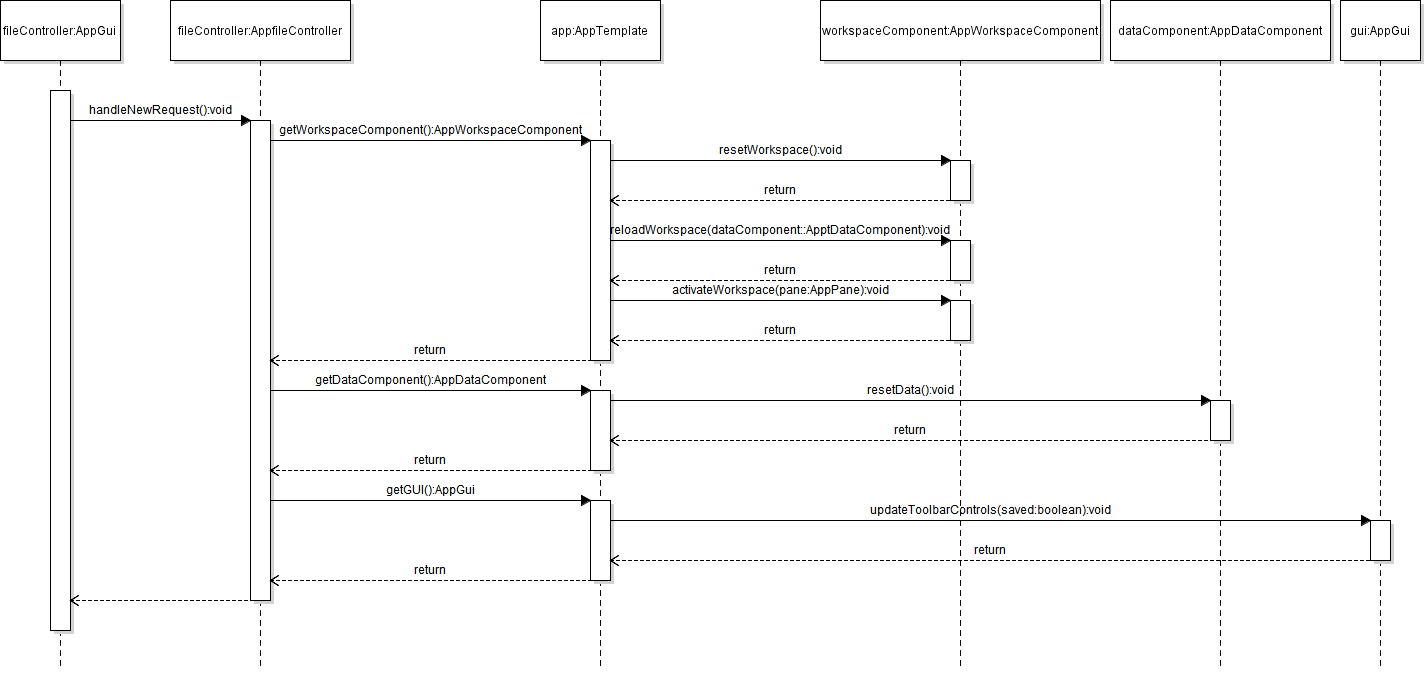
****

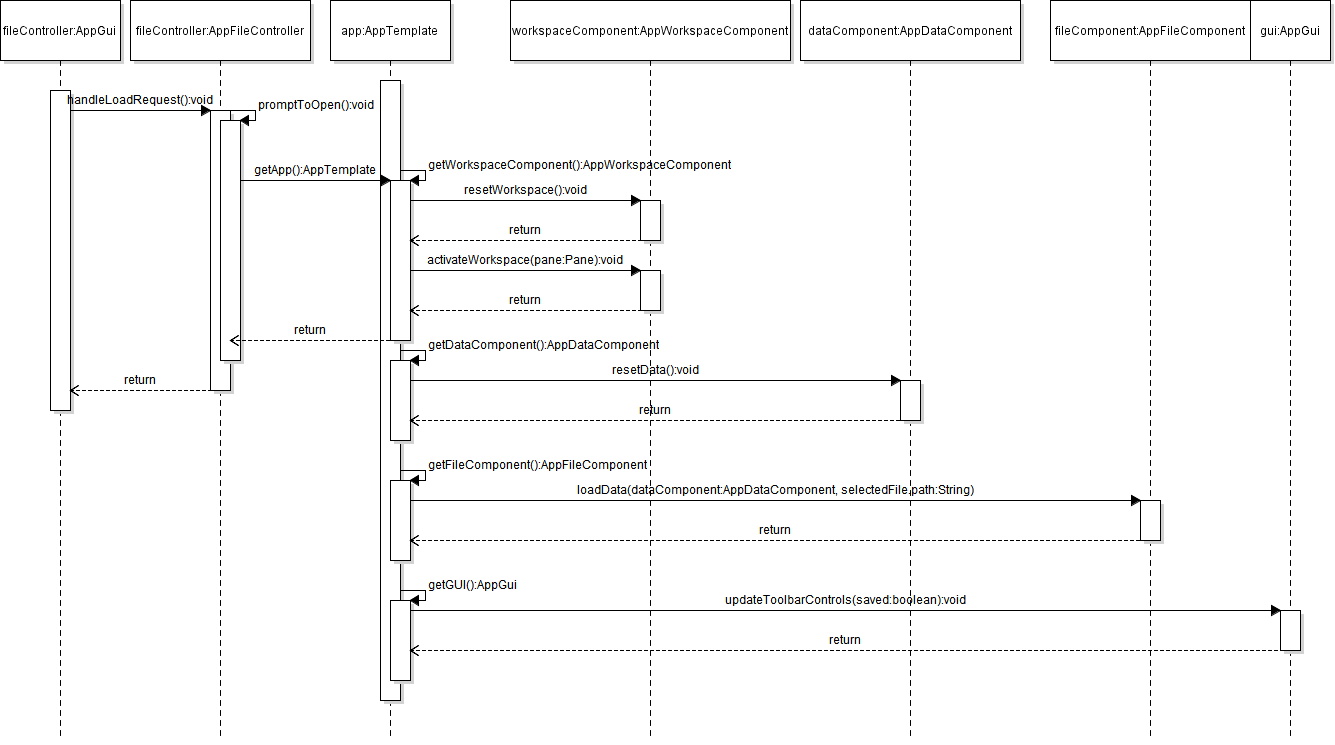
**Figure 4.1: Create New Map (2.1) UML Sequence Diagrams**

**Figure 4.2: Select Recent Map to Load (2.2)** **UML Sequence Diagrams**

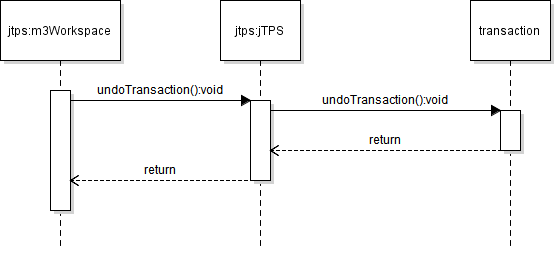
****

**Figure 4.3: Close Welcome Dialog (2.3) UML Sequence Diagrams**

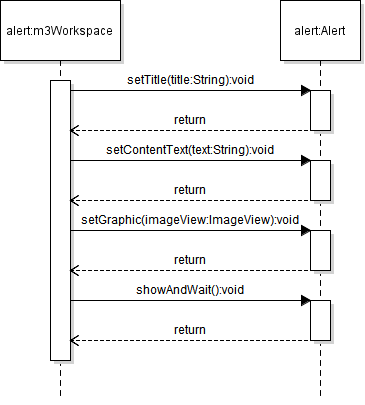
**Figure 4.4: Create New Map from the main UI (2.4) UML Sequence Diagrams**

****

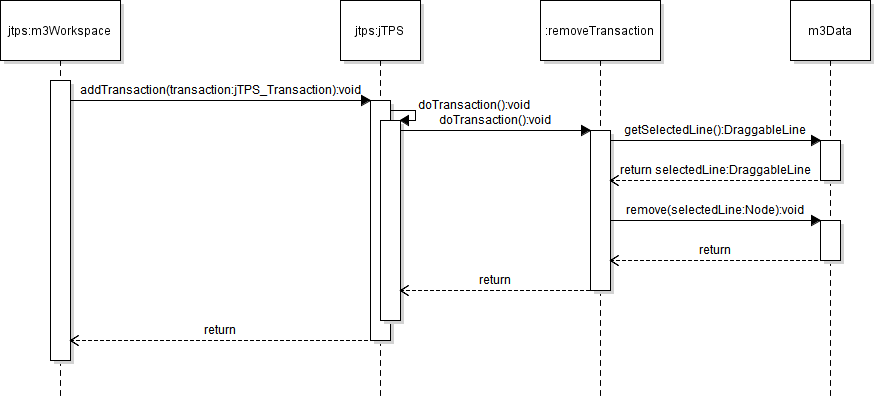
**Figure 4.5: Load Map from the main UI (2.5) UML Sequence Diagrams**

****

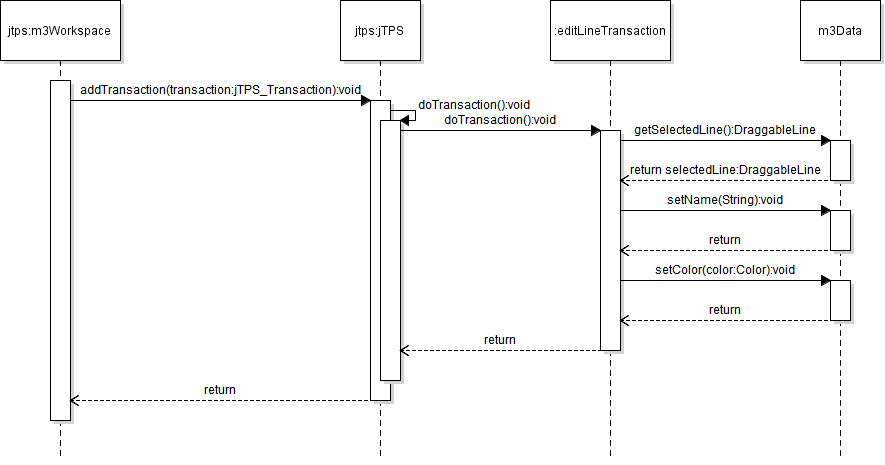
**Figure 4.6: Undo Edit (2.9) UML Sequence Diagrams**

****

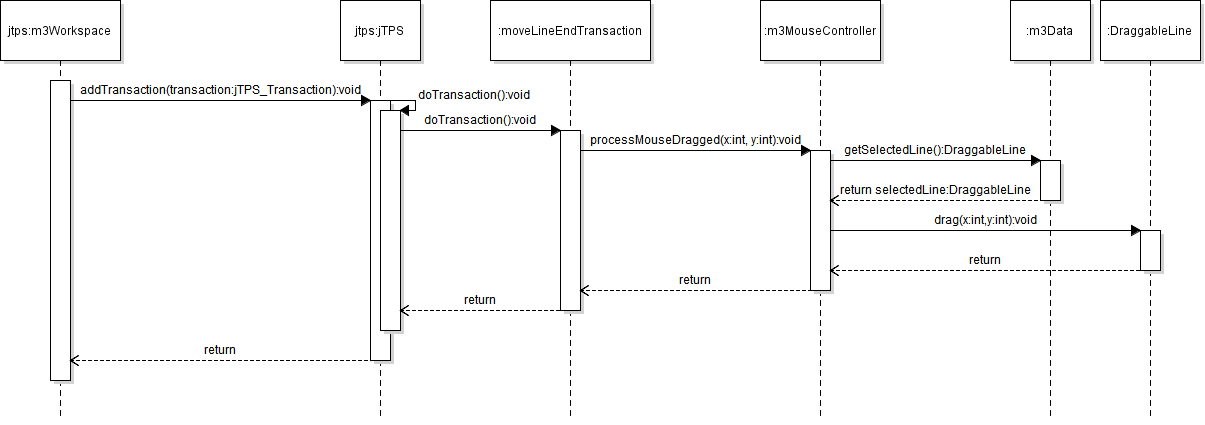
**Figure 4.7: Learn About Application (2.11) UML Sequence Diagrams**

****

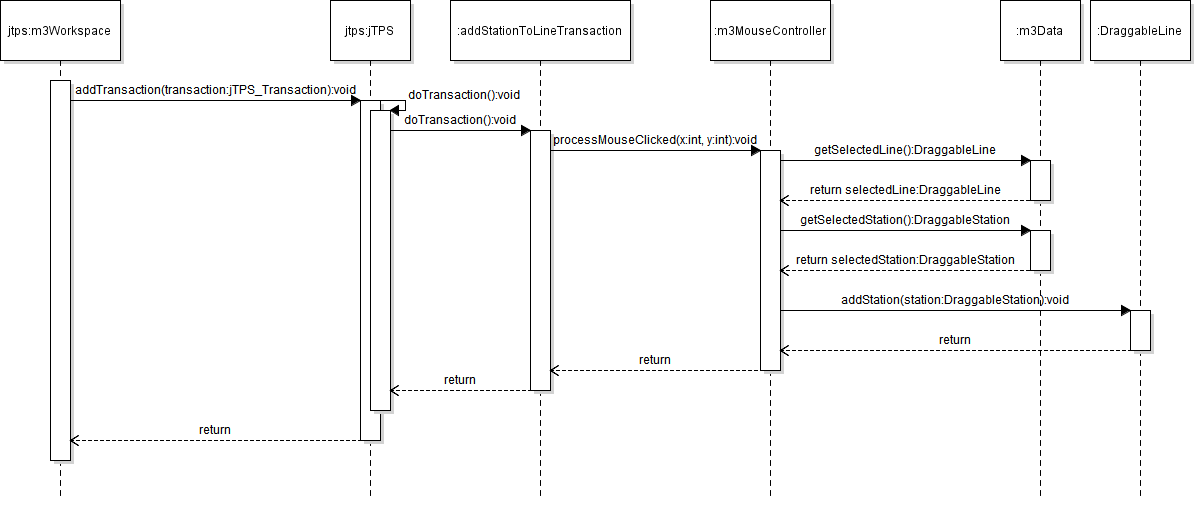
**Figure 4.8: Remove Line (2.13) UML Sequence Diagrams**

****

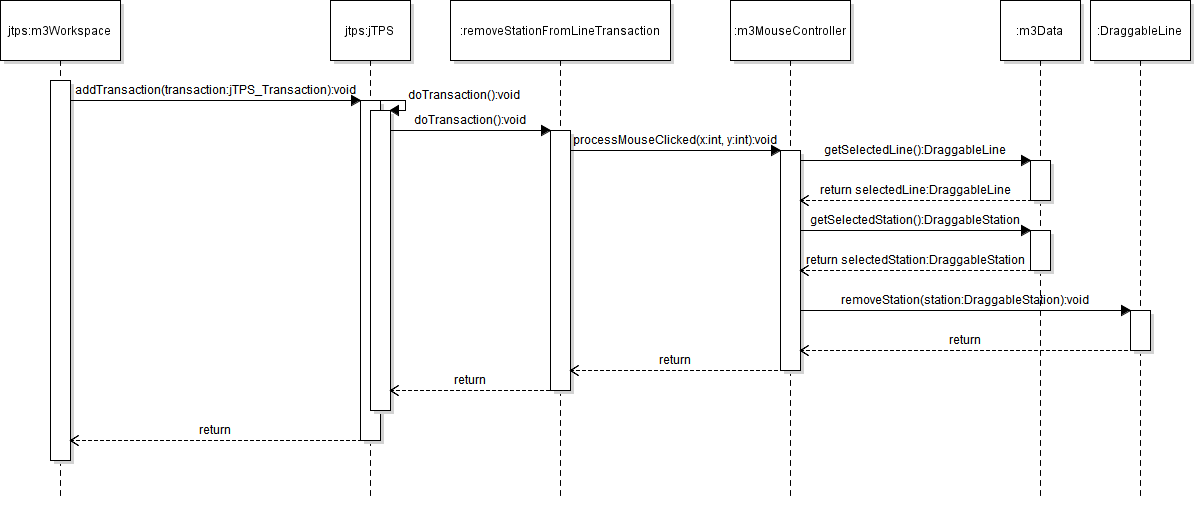
**Figure 4.9: Edit Line (2.14) UML Sequence Diagrams**

****

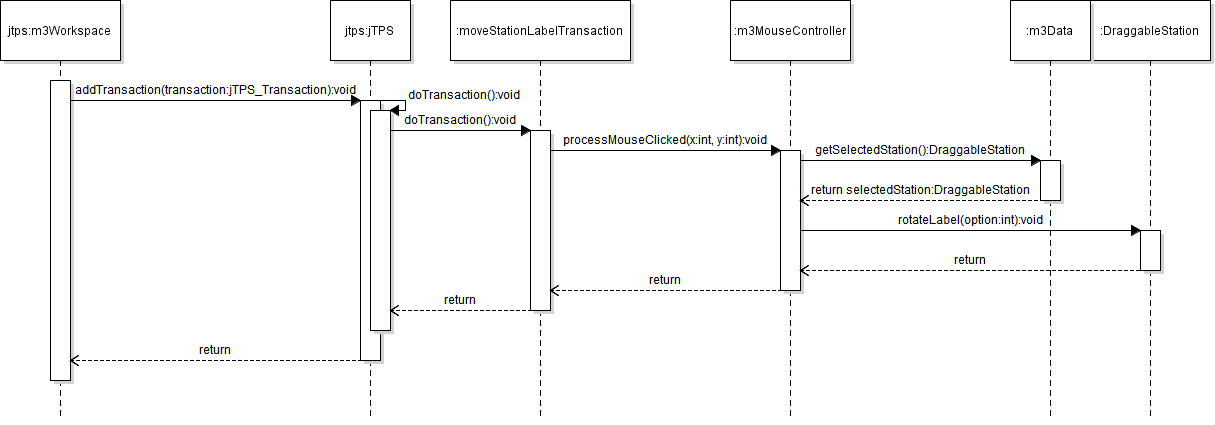
**Figure 4.10: Move Line End (2.15) UML Sequence Diagrams**

****

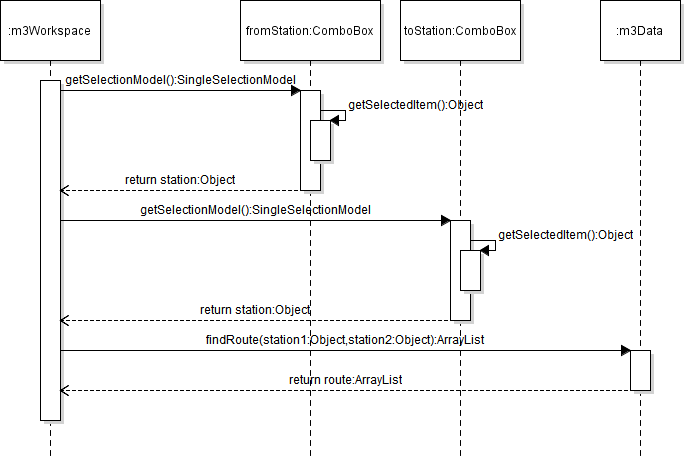
**Figure 4.11: Add Stations to Line (2.16) UML Sequence Diagrams**

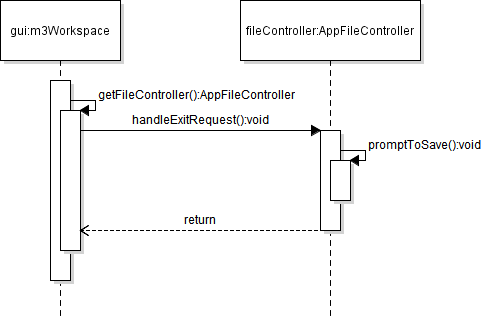
****

**Figure 4.12: Remove Stations from Line (2.17) UML Sequence Diagrams**

****

**Figure 4.13: Move Station Labels (2.23) UML Sequence Diagrams**

****

**Figure 4.14: Find Route (2.27) UML Sequence Diagrams**

**Figure 4.15: Exit Application (2.39) UML Sequence Diagrams**

**5. File Structure and Formats**

Note that the Desktop Java Framework and Properties Manager will be provided inside MetroMapMaker.jar, a Java ARchive file that will encapsulate the entire framework. This should be imported into the necessary project for the MetroMapMaker application and will be included in the deployment of a single, executable JAR file titled MetroMapMaker.jar. Note that all necessary data and art files must accompany this program. Figure 5.1 specifies the necessary file structure the launched application should use. Note that all necessary images should of course go in the image director, and the required xml files should go in the data director. All maps made from this application will be saved in the work director and the exported data will go in the export directory.

* MetroMapMaker

Screen Clipping MetroMapMaker

Screen Clipping images

data



Screen Clipping work

Screen Clipping export

**Figure 5.1: MetroMapMaker File Structure 6. Supporting Information**

Note that this document should serve as a reference for those implementing the code, so we’ll provide a table of contents to help quickly find important sections.

**6.1 Table of contents**

1. Introduction 2

1. Purpose 2
2. Scope 2
3. Definitions, acronyms, and abbreviations 2
4. References 3
5. Overview 3

2. Package-Level Design Viewpoint 4

1. The Metro Map Maker™ overview 4
2. Java API Usage 5
3. Java API Usage Descriptions 6

3. Class-Level Design Viewpoint 9

4. Method-Level Design Viewpoint 25

5. File Structure and Formats 32

6. Supporting Information 33

1. Table of contents 33
2. Appendixes 33

**6.2 Appendixes**

N/A