**The *Metro Map Maker* TM0**

**Software Design Description**

**Author:** Nicholas Shaw

Debugging EnterprisesTM

October, 2017

Version 1.0

**Abstract:** This document describes the software design for The Metro Map Maker, a tool that allows individuals to create and edit subway maps.

**Based on IEEE Std 1016TM-2017 document format**

Copyright © 2017 Debugging Enterprises, which is a made up company and doesn’t really own The Metro Map Maker.

*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*** subway map builder application. Note that this document format is based on the IEEE Standard 1016-2017 recommendation for software design.

**1.1 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.

**1.2 Scope**

The ***Metro Map Maker*** will be a tool built using a common framework named the Desktop Java Framework which can be used in the future for other possible applications, and as such may be modified to suit the specific use cases for the ***Metro Map Maker***. With that in mind, this document will contain information on the modification of this framework as well as the creation of the ***Metro Map Maker*** as well. Note that Java is the target language for this software design.

**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 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.

**GUI** – The GUI or Graphical User Interface, are what is displayed to the user that allows them to manipulate and use the application in the form of icons and graphics instead of the use of text-based interfaces.

**Desktop Java Framework** – The software framework to be used to create the ***Metro Map Maker*** application, and can be used to create other JavaFX-based desktop applications. This framework contains the base classes for implementing any desktop Java applications as well as certain Graphical User Interface (GUI) controls and features.

**Properties Manager** – This library will be used by our ***Metro Map Maker*** to help manager the property files for the application, such as language properties and image file locations.

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

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

**1.4 References**

**IEEE Std 830TM-1998 (R2017) –** IEEE Standard for Information Technology – Systems Design – Software

Design Descriptions

**The Metro Map MakerTM0 SRS** – Debugging Enterprises’ Software Requirements Specification for the Metro Map Maker application.

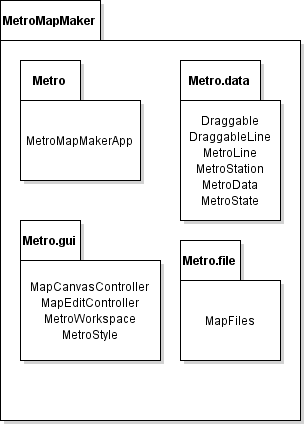
**1.5 Overview**

This Software Design Description document provides a working design for the ***Metro Map Maker*** software application as described in the Metro Map Maker Software Requirements Specification. Note that all parties in the implementation stage must agree upon all connections between components before proceeding with the implementation stage. Section 2 of this document will provide the Package-Level Viewpoint, specifying the packages and frameworks. 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.

**2. Package-Level Design Viewpoint**

**2.1 Metro Map Maker Overview**

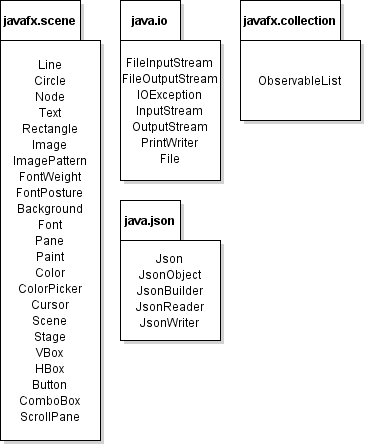
Figure 2.1 specifies all the component classes of our application that will be developed separated into packages.



**Figure 2.1: Design Packages Overview**

**2.2 Java API Usage**

As our application will be developed using the Java programing language and use JavaFX, we will make use of the classes specified in Figure 2.2.



**Figure 2.2: Java API Classes and Packages to be used**

**2.3 Java API Usage Descriptions**

Tables 2.1 – 2.4 below will summarize how each of these classes will be used.

|  |  |
| --- | --- |
| Class/Interface | Use |
| Line | Used to create lines for the metro line objects |
| Circle | Used to display stations on metro lines |
| Node | A parent object of all stage objects that is used as a reference |
| Text | Used to display text on the user interface |
| Rectangle | For creating the paths for the metro lines |
| Image | For creating logos, background images and any other images used |
| ImagePattern | Contains Images to be displayed within the Pane |
| FontWeight | Used to make labels and text bold |
| FontPosture | Used to make labels and text italics |
| Font | Used to change label and text color, size and font family |
| Background | Used to set background images and colors for the workspace |
| Pane | Contains any objects, buttons, sliders etc on the user interface |
| Scene | Contains all Panes on the user interface to display |
| Stage | Contains the pane and is used for the main application window |
| Color | Used for font color, metro line color and station color |
| Paint | Used to set background colors and taken from ColorPicker |
| ColorPicker | Used to pick colors for metro lines, stations, font and backgrounds |
| Cursor | Used to retrieve information from the cursor location to be used with moving objects |
| VBox | Used to contain buttons and user interface elements in a vertical array to display |
| HBox | Used to contain buttons and user interface elements in a horizontal array to display |
| Button | Used to create buttons on the user interface |
| ComboBox | Used to create an expandable list to be used for font sizes, font family, metro line list etc |

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

|  |  |
| --- | --- |
| Class/Interface | Use |
| File | Used to reference files to be loaded such as setting and recent works files |
| InputStream | Used to read input from the file that is being referenced so assign settings |
| OutputStream | Used to write to a file to change setting and to keep track of recent files |
| IOException | An exception that is thrown when there is an error in file loading |
| FileInputStream | Used to read from files, a more specialized version of InputStream |
| FileOutputStream | Used to write to files, a more specialized version of OutputStream |
| PrintWriter | Used to write to files in a similar way to FileOutputStream |

**Table 2.2: Uses for classes in the Java API’s java.io package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| Json | A file format that will be used to save out maps to be edited later |
| JsonObject | Used to create Json Objects that will be saved |
| JsonBuilder | Used to build Json files that will be used to save work |
| JsonReader | Used to read Json files that will be used to load work |
| JsonWriter | Used to write to Json files that will be used to update existing work |

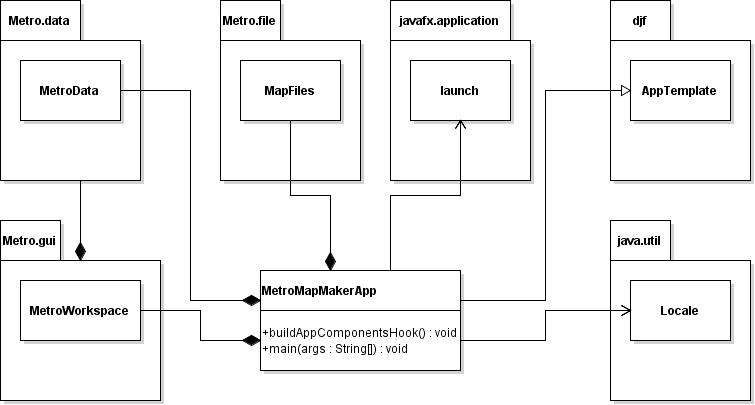
**Table 2.3: Uses for classes in the Java API’s java.json package**

|  |  |
| --- | --- |
| Class/Interface | Use |
| ObservableList | An observable list of nodes that will be used to access scene elements |

**Table 2.3: Uses for classes in the JavaFX API’s javafx.collection package**

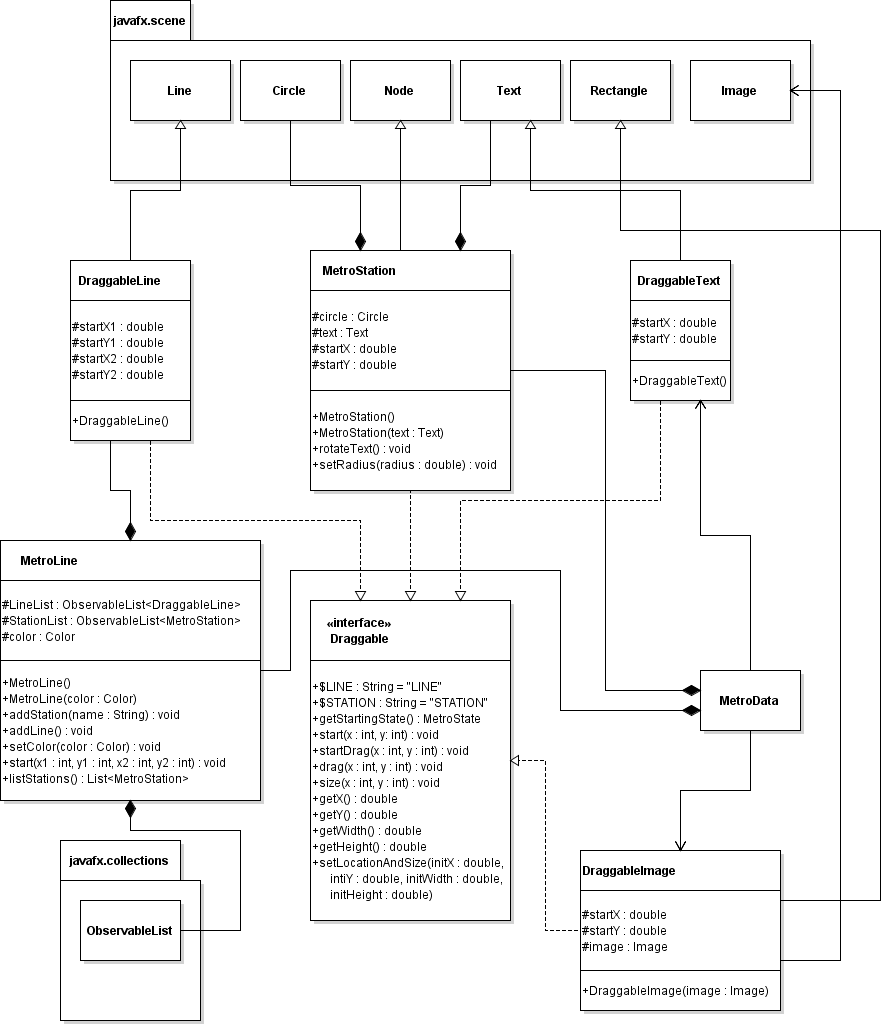
**3. Class-Level Design Viewpoint**

For this application we will be utilizing the DesktopJavaFramework on which we can build our map maker application, as it provides the necessary base functionality for a desktop application. Through the use of DJF, we can create an application file and extend the different components of the DJF, such as dataComponent and workspaceComponent, to create a cohesive application. This works much more effectively than other possible implementations, as it creates a very clear workflow throughout the entirety of the application and makes for a smooth experience not just for the user, but also any developer working on the application. The following UML diagram (Figure 3.1) shows the interaction of the different packages with our main package, the MetroMapMakerApp package.

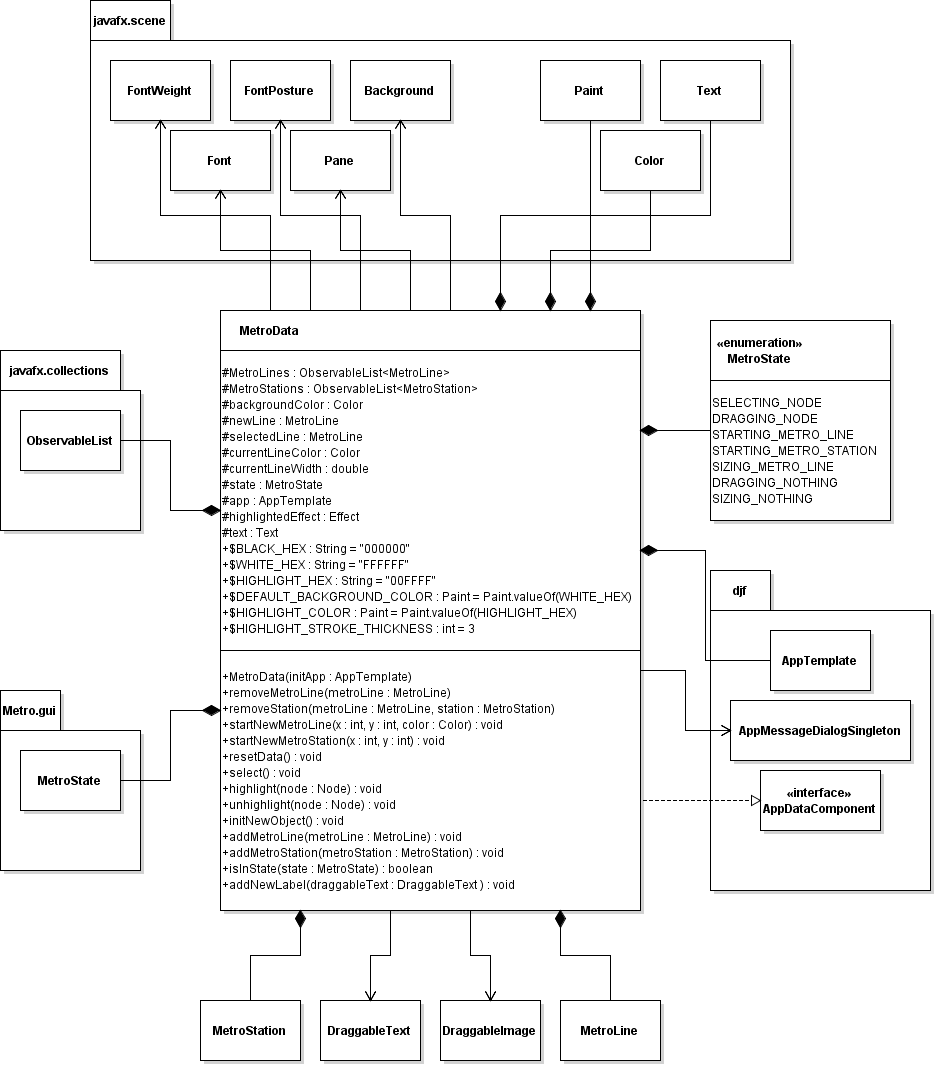


**Figure 3.1: Metro Map Maker Overview UML Diagram**

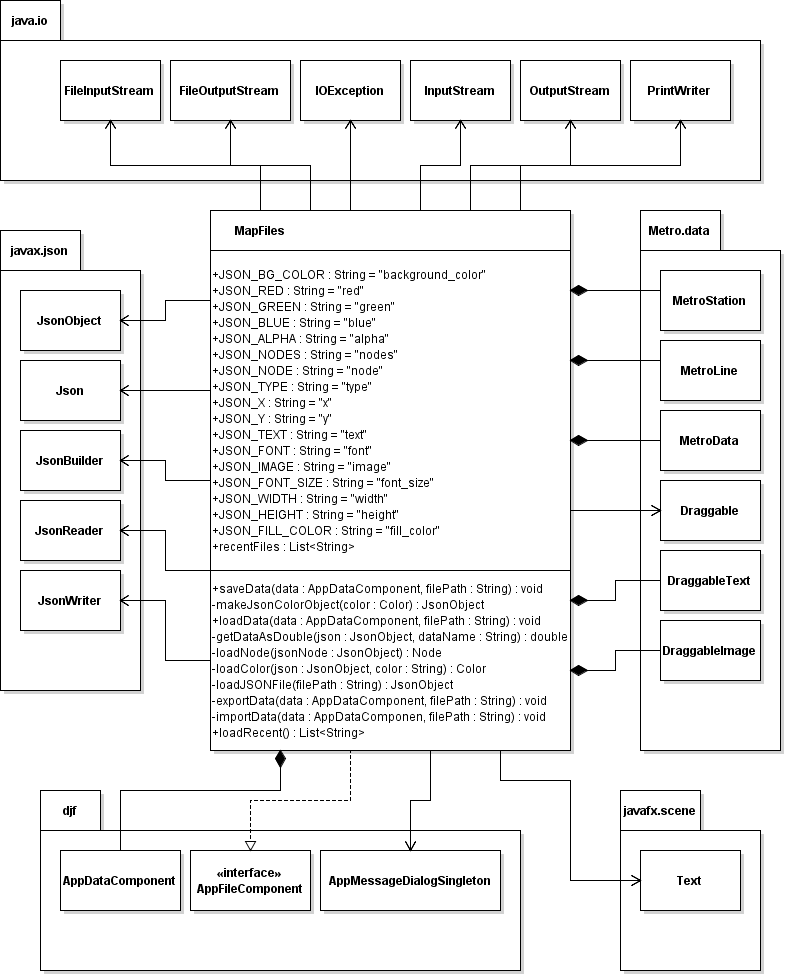
The remaining UML Class Diagrams will show the interactions between the individual classes within their respective packages.



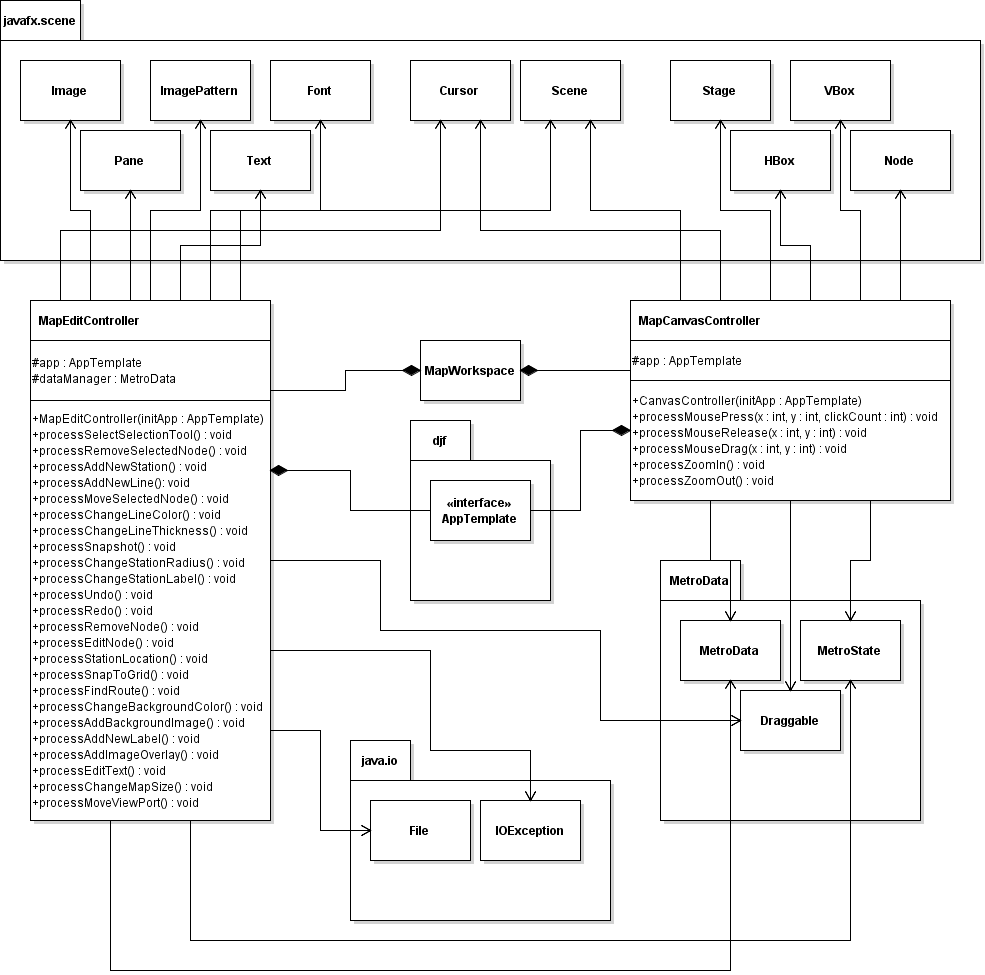
**Figure 3.2: Draggable Classes within the Metro.data Package**

****

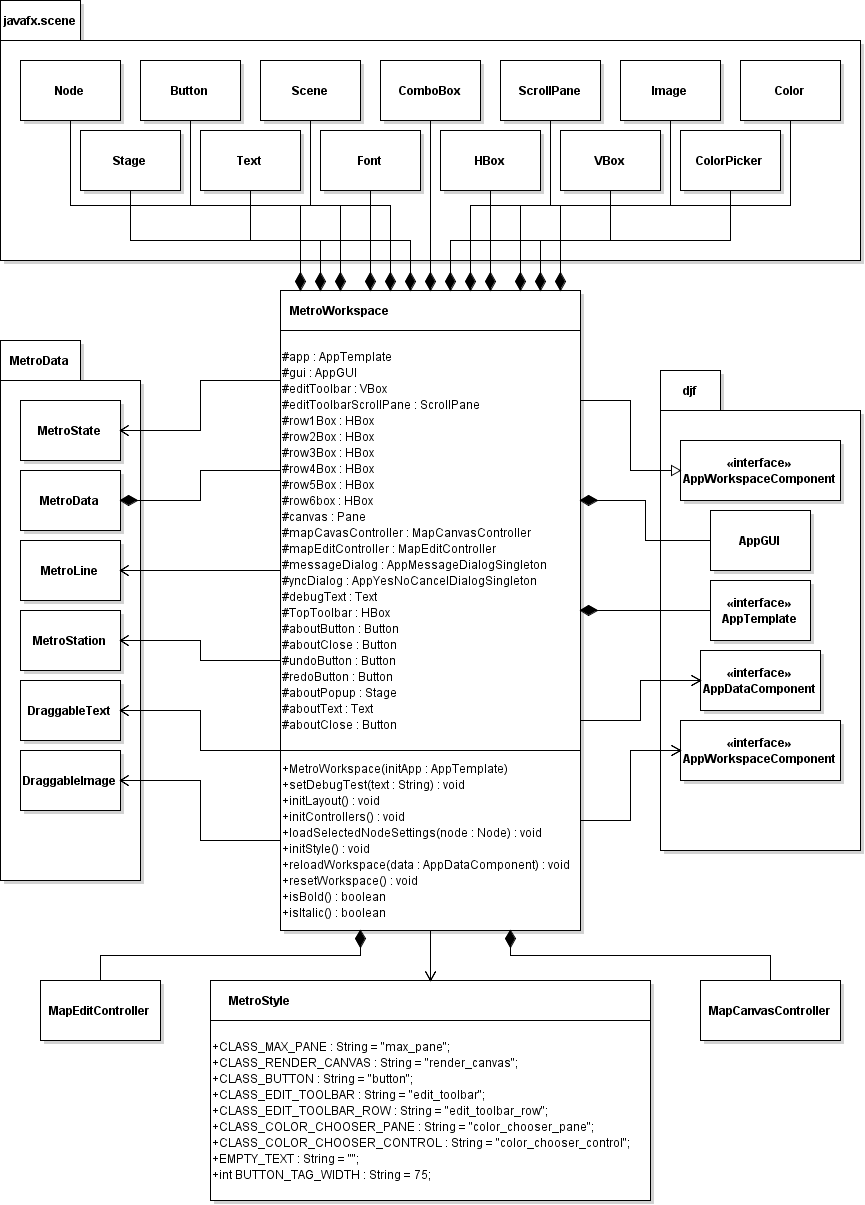
**Figure 3.3: MetroData Class within the Metro.data Package, and it’s interaction with the Draggable Classes**

****

**Figure 3.4: MapFiles Class within the Metro.file Package**

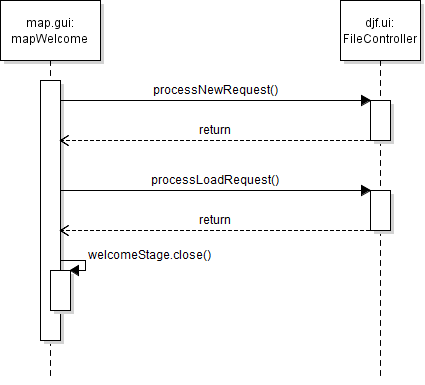
****

**Figure 3.5: Controller Classes within the Metro.gui Package**

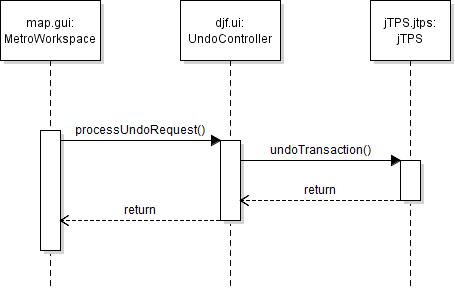
****

**Figure 3.6: The MetroWorkspace and MetroStyle Classes within the Metro.gui Package**

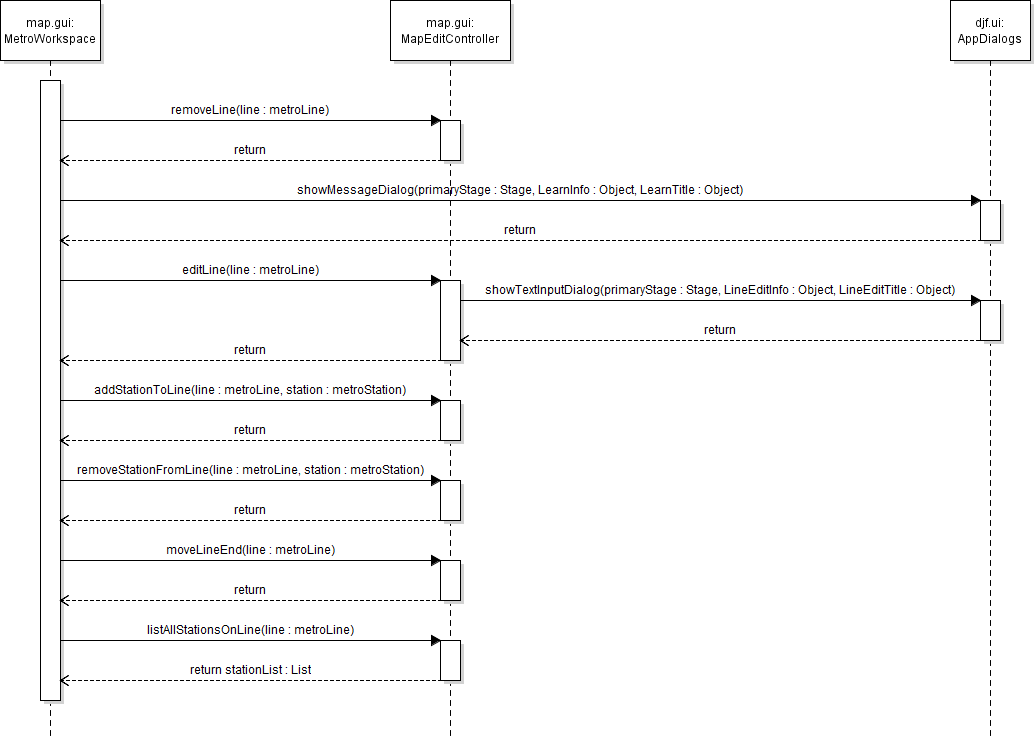
**4. Method-Level Design Viewpoint**

****

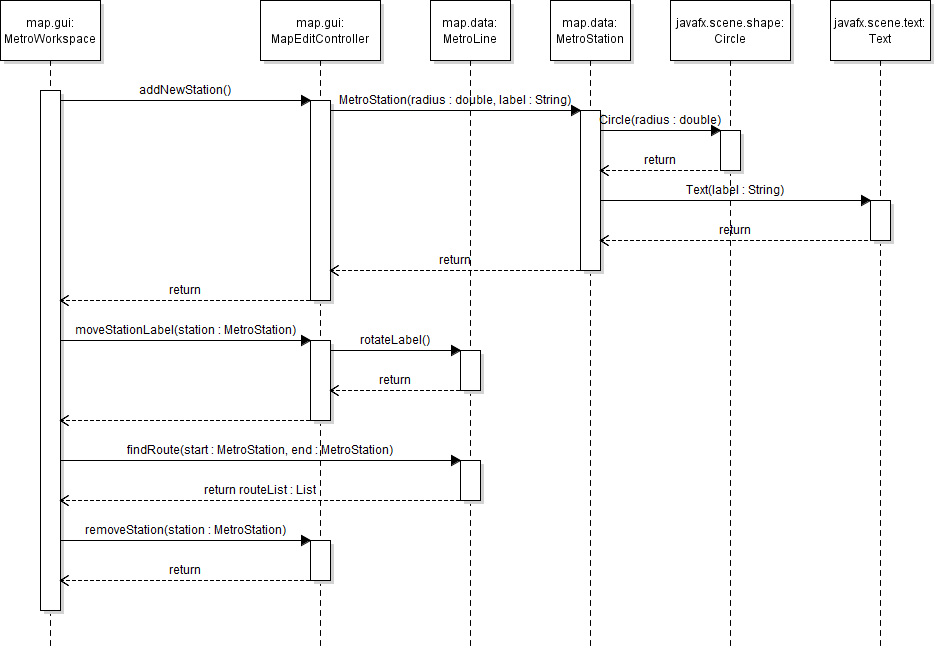
**Figure 4.1: Welcome Dialog UML Sequence (2.1, 2.2, 2.3)**

****

**Figure 4.2: Undo Transaction UML Sequence (2.9)**



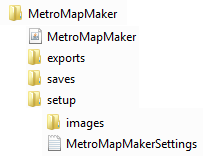
**Figure 4.3: Line Edit UML Sequence (2.11, 2.13, 2.14, 2.15, 2.16, 2.17, 2.18)**

****

**Figure 4.4: MetroLine and MetroStation Edit UML Sequence (2.20, 2.21, 2.23, 2.27)**

**5. File Structure and Formats**

Note that the DesktopJavaFramework will be held within a JAR file, a Java ARchive file that will encapsulate the entire framework. This should be imported into the necessary project for the Metro Map Maker application and will be included in the deployment of a single, executable JAR file titled Zombiquarium.jar. Note that all necessary data needs to accompany this program. Figure 5.1 specifies the necessary file structure the launched application should use.



**Figure 5.1: Metro Map Maker File Structure**

Our map maker application will utilize text files to keep track of recently opened and edited maps for ease of use for the user. This data will be stored as a plain text filePath within a “data” folder in our base Metro Map Maker folder

The format of our saved files will be Json. These Json files will contain encoded data for images, if used, encoded objects for each metro line, containing data for size, color, and a list of metro stations within the metro line object, each with its own size, color and label data.

**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 | 2-3 |
| 5. Overview | 3 |
| 2. Package-Level Design Viewpoint | 4 |
| 1. Metro Map Maker Overview | 4 |
| 2. Java API Usage | 5 |
| 3. Java API Usage Descriptions | 5-7 |
| 3. Class-Level Design Viewpoint | 5-13 |
| 4. Method-Level Design Viewpoint | 14-16 |
| 5. File Structure and Formats | 17 |
| 6. Supporting Information | 18 |
| 1. Table of Contents | 18 |
| 2. Appendixes | 18 |

**6.2 Appendixes**

N/A