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. 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

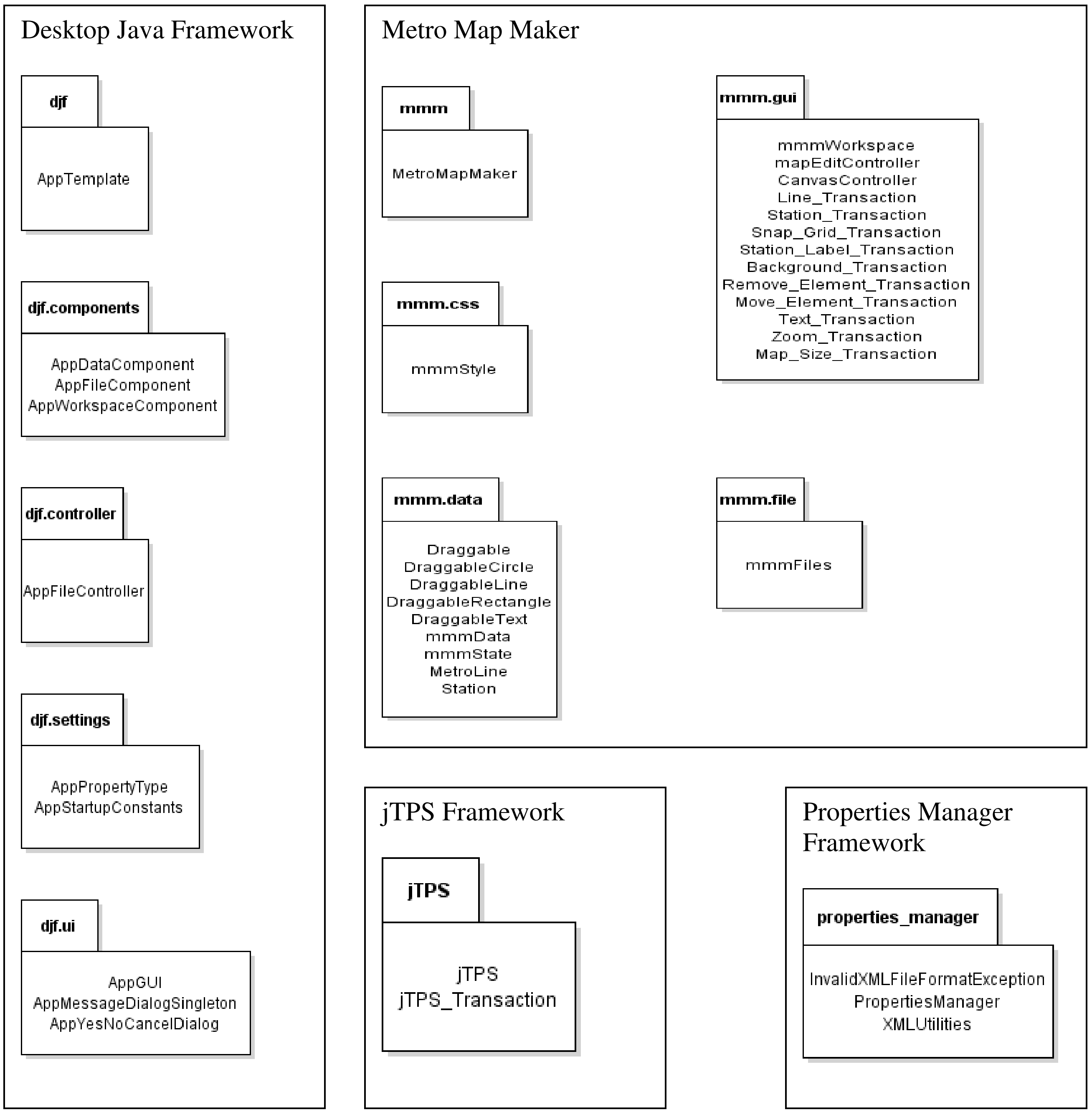


Figure 2.1: Design Packages Overview

2.2 Java API Usage

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

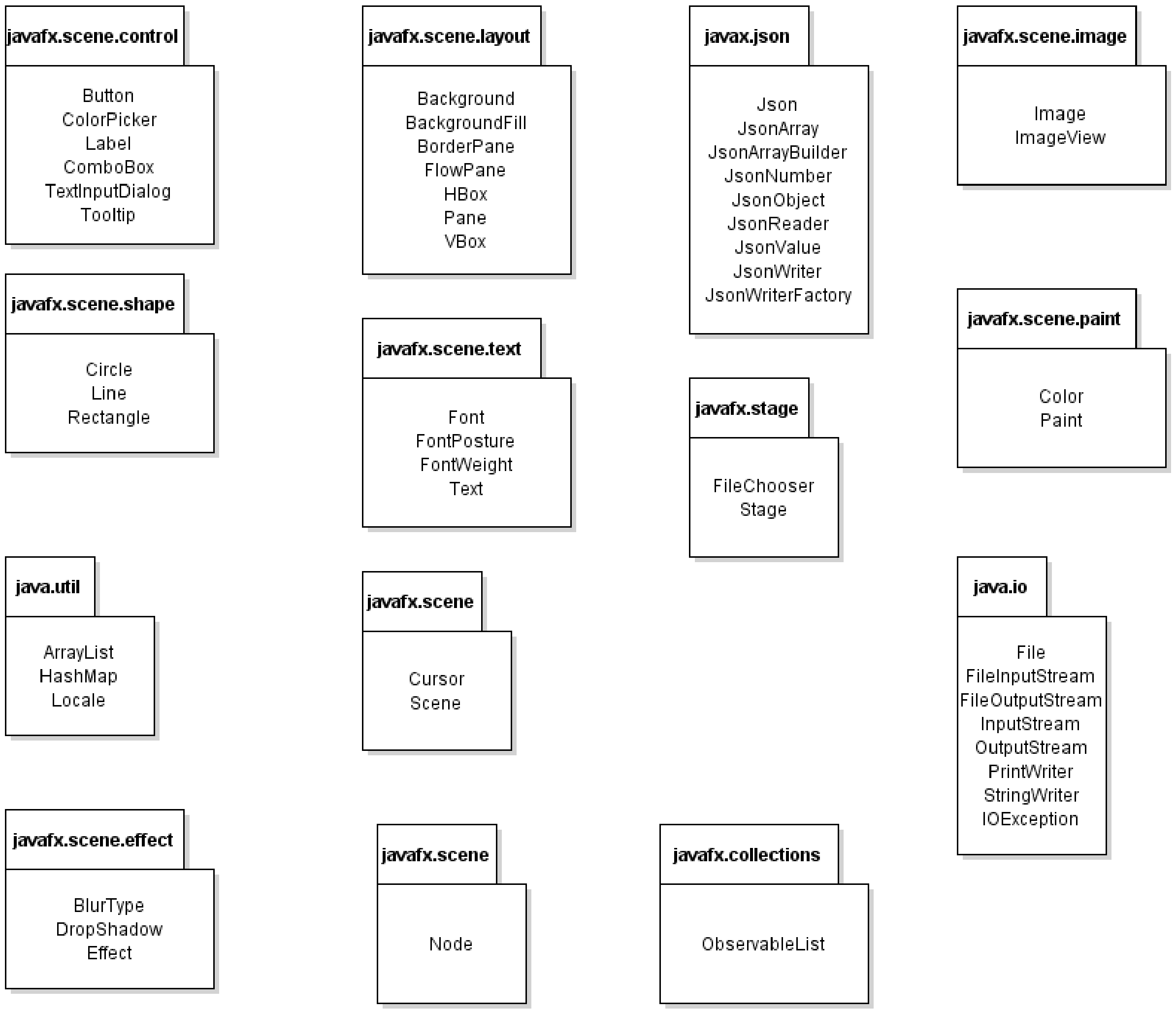


Figure 2.2: Java API Classes and Packages To Be Used

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

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.

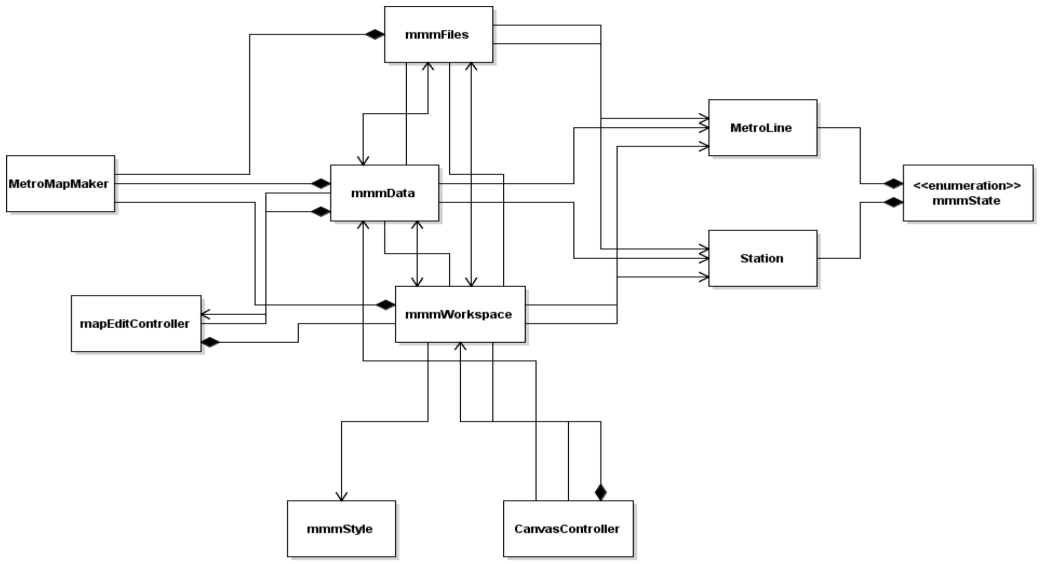


Figure 3.1: Metro Map Maker Overview UML Diagram

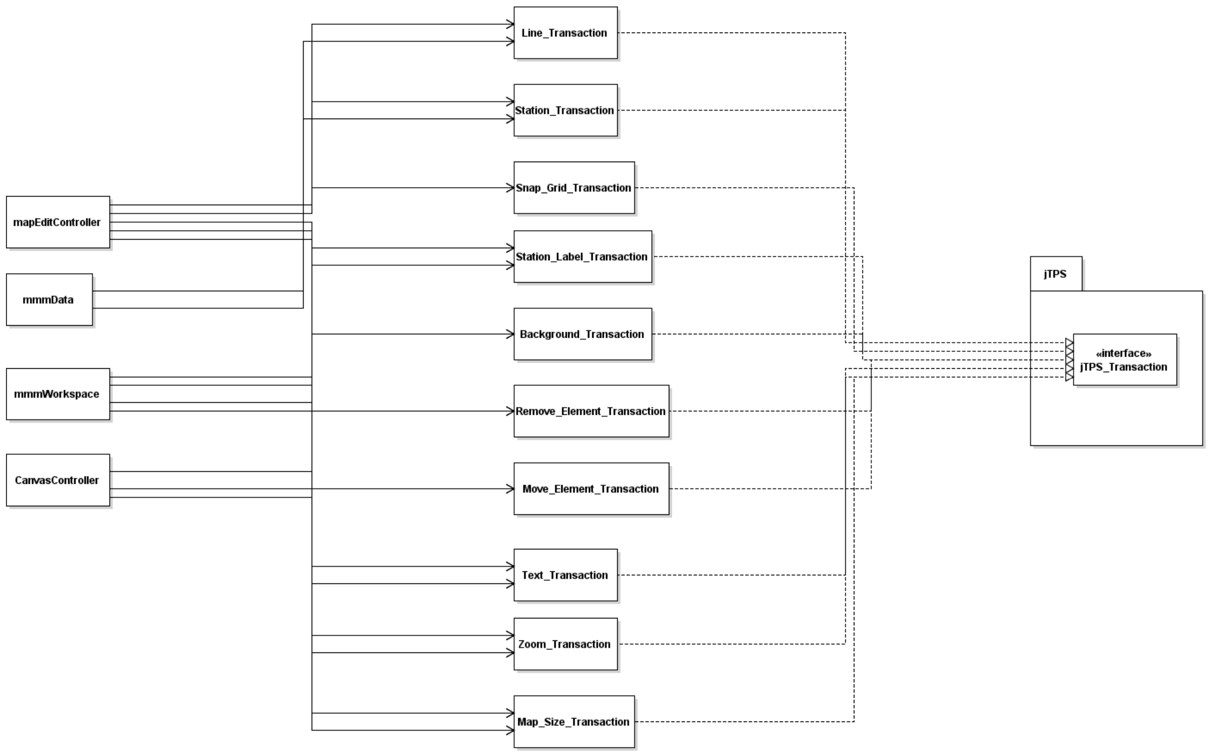


Figure 3.2: Transactions Overview UML Diagram

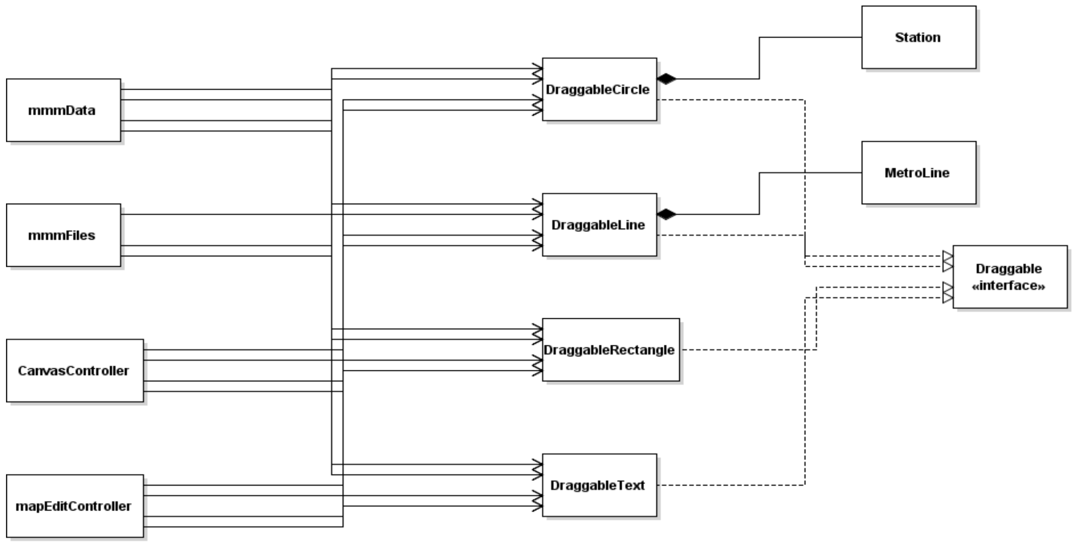
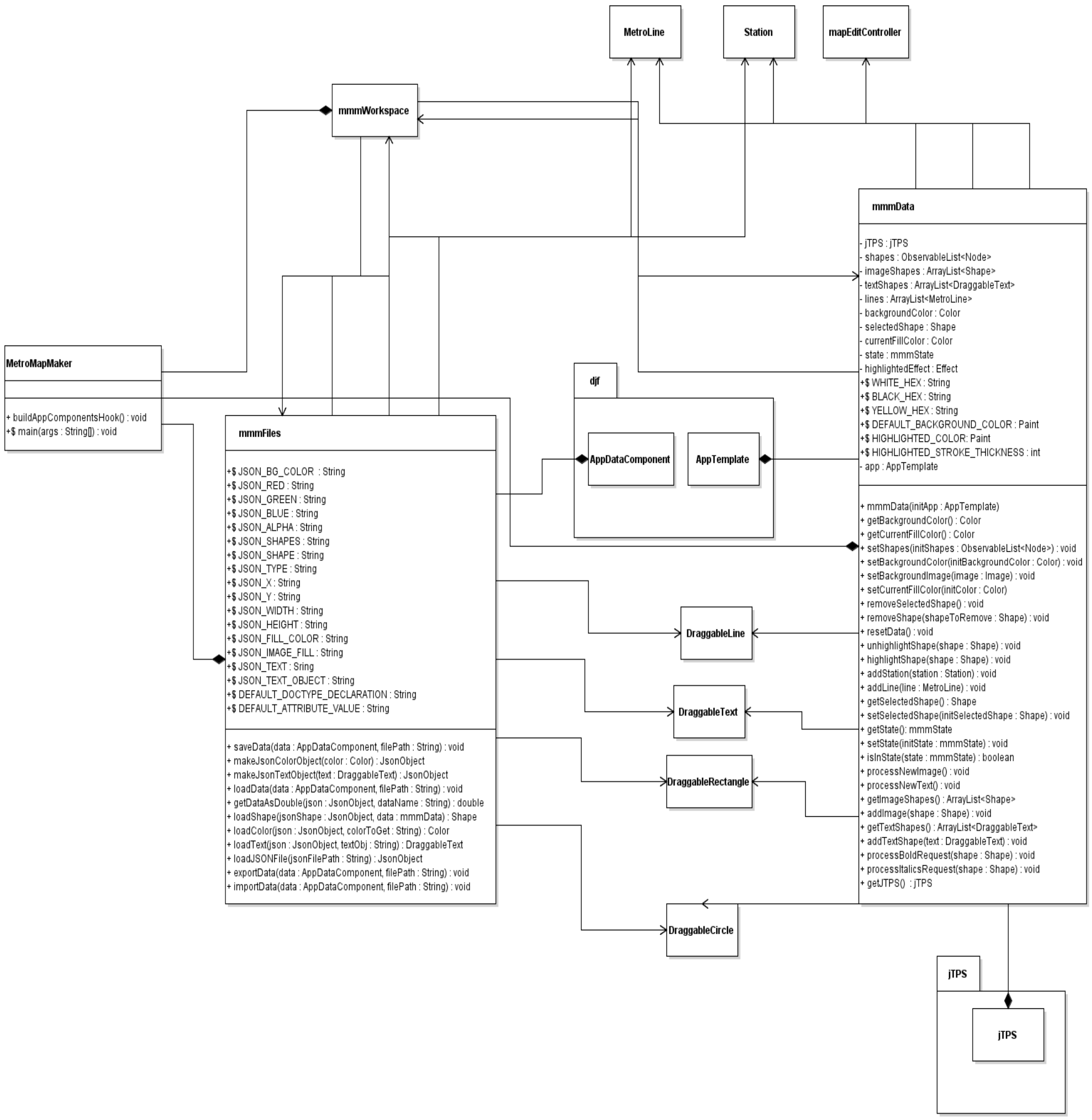


Figure 3.3: Draggable Overview UML Diagram



MetroLine
Station
mapEditController
mmmWorkspace
mmmData
- jTPS : jTPS
- shapes : ObservableList<Node>
- imageShapes : ArrayList<Shape>
- textShapes : ArrayList<DraggableText>
- lines : ArrayList<MetroLine>
- backgroundColor : Color
- selectedShape : Shape
- currentFillColor : Color
- state : mmmState
- highlightedEffect : Effect
+$ WHITE_HEX : String
+$ BLACK_HEX : String
+$ YELLOW_HEX : String
+$ DEFAULT_BACKGROUND_COLOR : Paint
+$ HIGHLIGHTED_COLOR: Paint
+$ HIGHLIGHTED_STROKE_THICKNESS : int
- app : AppTemplate
+ mmmData(initApp : AppTemplate)
+ getBackgroundColor() : Color
+ getCurrentFillColor() : Color
+ setShapes(initShapes : ObservableList<Node>) : void
+ setBackgroundColor(initBackgroundColor : Color) : void
+ setBackgroundImage(image : Image) : void
+ setCurrentFillColor(initColor : Color)
+ removeSelectedShape() : void
+ removeShape(shapeToRemove : Shape) : void
+ resetData() : void
+ unhighlightShape(shape : Shape) : void
+ highlightShape(shape : Shape) : void
+ addStation(station : Station) : void
+ addLine(line : MetroLine) : void
+ getSelectedShape() : Shape
+ setSelectedShape(initSelectedShape : Shape) : void
+ getState(): mmmState
+ setState(initState : mmmState) : void
+ isInState(state : mmmState) : boolean
+ processNewImage() : void
+ processNewText() : void
+ getImageShapes() : ArrayList<Shape>
+ addImage(shape : Shape) : void
+ getTextShapes() : ArrayList<DraggableText>
+ addTextShape(text : DraggableText) : void
+ processBoldRequest(shape : Shape) : void
+ processItalicsRequest(shape : Shape) : void
+ getJTPS() : jTPS
MetroMapMaker
+ buildAppComponentsHook() : void
+$ main(args : String[]) : void
mmmFiles
+$ JSON_BG_COLOR : String
+$ JSON_RED : String
+$ JSON_GREEN : String
+$ JSON_BLUE : String
+$ JSON_ALPHA : String
+$ JSON_SHAPES : String
+$ JSON_SHAPE : String
+$ JSON_TYPE : String
+$ JSON_X : String
+$ JSON_Y : String
+$ JSON_WIDTH : String
+$ JSON_HEIGHT : String
+$ JSON_FILL_COLOR : String
+$ JSON_IMAGE_FILL : String
+$ JSON_TEXT : String
+$ JSON_TEXT_OBJECT : String
+$ DEFAULT_DOCTYPE_DECLARATION : String
+$ DEFAULT_ATTRIBUTE_VALUE : String
+ saveData(data : AppDataComponent, filePath : String) : void
+ makeJsonColorObject(color : Color) : JsonObject
+ makeJsonTextObject(text : DraggableText) : JsonObject
+ loadData(data : AppDataComponent, filePath : String) : void
+ getDataAsDouble(json : JsonObject, dataName : String) : double
+ loadShape(jsonShape : JsonObject, data : mmmData) : Shape
+ loadColor(json : JsonObject, colorToGet : String) : Color
+ loadText(json : JsonObject, textObj : String) : DraggableText
+ loadJSONFile(jsonFilePath : String) : JsonObject
+ exportData(data : AppDataComponent, filePath : String) : void
+ importData(data : AppDataComponent, filePath : String) : void
djf
AppDataComponent
AppTemplate
DraggableLine
DraggableText
DraggableRectangle
DraggableCircle
jTPS
jTPS

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

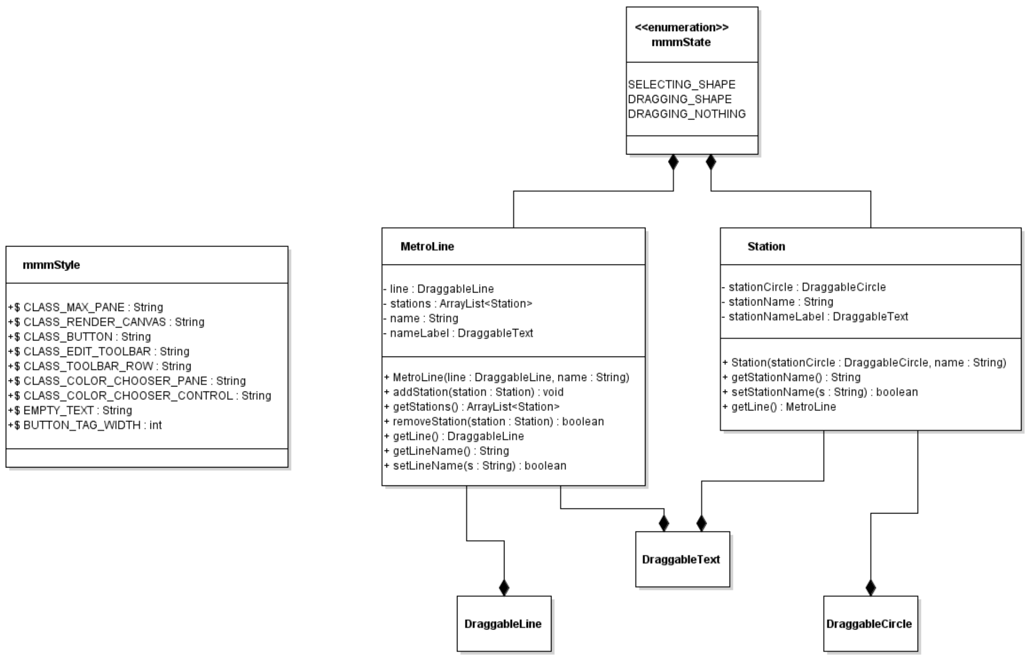

<<enumeration>>
mmmState
SELECTING_SHAPE
DRAGGING_SHAPE
DRAGGING_NOTHING
mmmStyle
+$ CLASS_MAX_PANE : String
+$ CLASS_RENDER_CANVAS : String
+$ CLASS_BUTTON : String
+$ CLASS_EDIT_TOOLBAR : String
+$ CLASS_TOOLBAR_ROW : String
+$ CLASS_COLOR_CHOOSER_PANE : String
+$ CLASS_COLOR_CHOOSER_CONTROL : String
+$ EMPTY_TEXT : String
+$ BUTTON_TAG_WIDTH : int
MetroLine
- line : DraggableLine
- stations : ArrayList<Station>
- name : String
- nameLabel : DraggableText
+ MetroLine(line : DraggableLine, name : String)
+ addStation(station : Station) : void
+ getStations() : ArrayList<Station>
+ removeStation(station : Station) : boolean
+ getLine() : DraggableLine
+ getLineName() : String
+ setLineName(s : String) : boolean
Station
- stationCircle : DraggableCircle
- stationName : String
- stationNameLabel : DraggableText
+ Station(stationCircle : DraggableCircle, name : String)
+ getStationName() : String
+ setStationName(s : String) : boolean
+ getLine() : MetroLine
DraggableText
DraggableLine
DraggableCircle

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

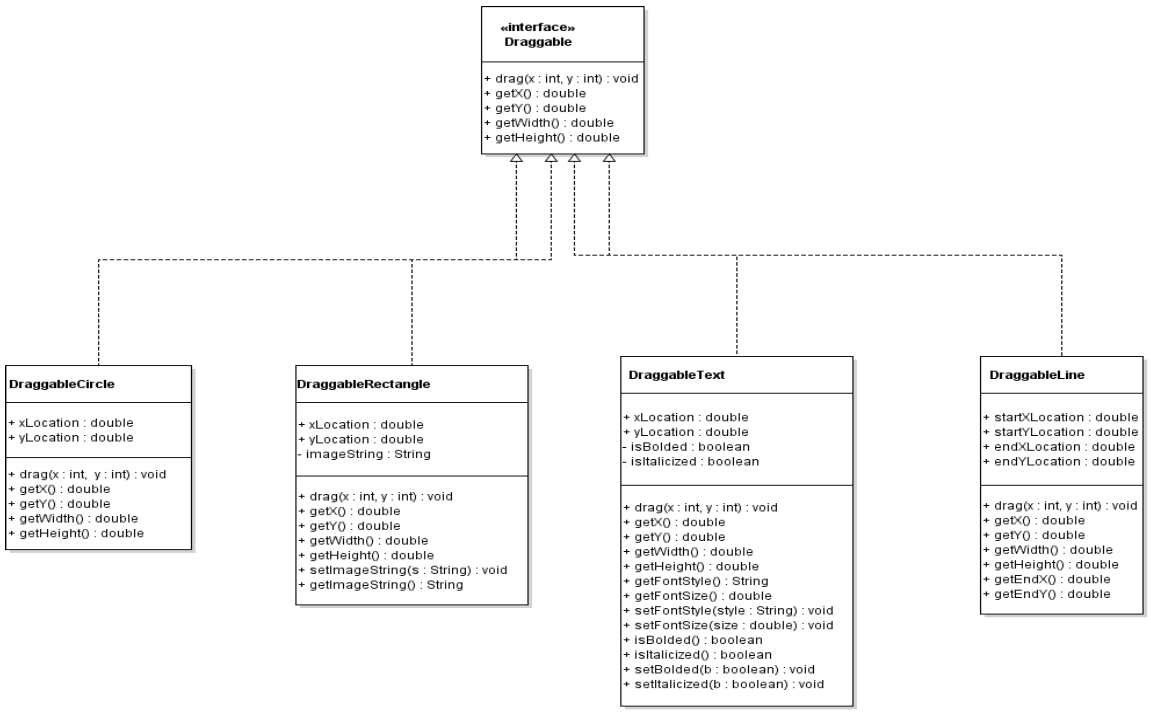


Figure 3.6: Detailed Draggable UML Class Diagrams

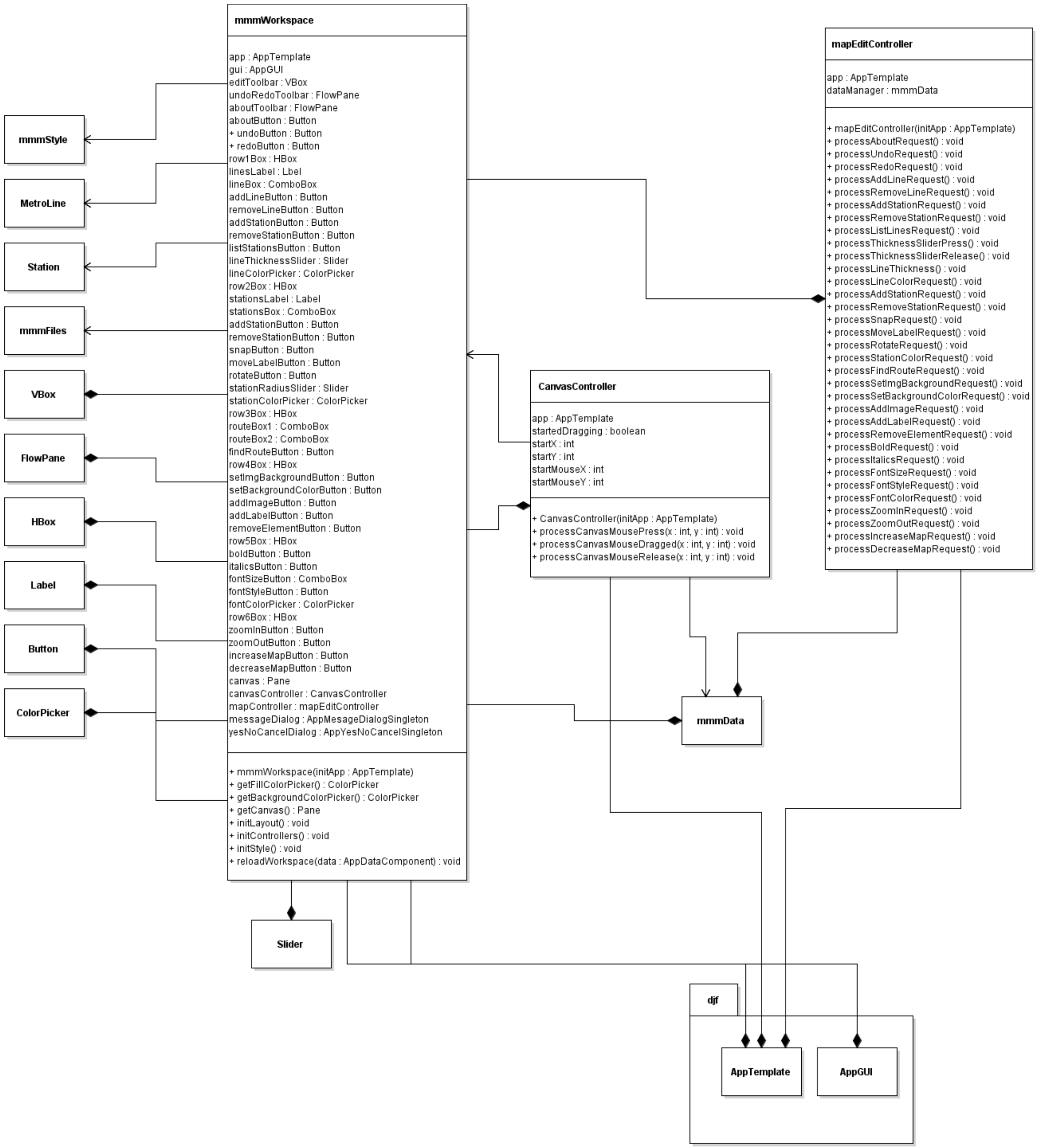

mmmWorkspace
app : AppTemplate
gui : AppGUI
editToolbar : VBox
undoRedoToolbar : FlowPane
aboutToolbar : FlowPane
aboutButton : Button
+ undoButton : Button
+ redoButton : Button
row1Box : HBox
linesLabel : Lbel
lineBox : ComboBox
addLineButton : Button
removeLineButton : Button
addStationButton : Button
removeStationButton : Button
listStationsButton : Button
lineThicknessSlider : Slider
lineColorPicker : ColorPicker
row2Box : HBox
stationsLabel : Label
stationsBox : ComboBox
addStationButton : Button
removeStationButton : Button
snapButton : Button
moveLabelButton : Button
rotateButton : Button
stationRadiusSlider : Slider
stationColorPicker : ColorPicker
row3Box : HBox
routeBox1 : ComboBox
routeBox2 : ComboBox
findRouteButton : Button
row4Box : HBox
setImgBackgroundButton : Button
setBackgroundColorButton : Button
addImageButton : Button
addLabelButton : Button
removeElementButton : Button
row5Box : HBox
boldButton : Button
italicsButton : Button
fontSizeButton : ComboBox
fontStyleButton : Button
fontColorPicker : ColorPicker
row6Box : HBox
zoomInButton : Button
zoomOutButton : Button
increaseMapButton : Button
decreaseMapButton : Button
canvas : Pane
canvasController : CanvasController
mapController : mapEditController
messageDialog : AppMesageDialogSingleton
yesNoCancelDialog : AppYesNoCancelSingleton
+ mmmWorkspace(initApp : AppTemplate)
+ getFillColorPicker() : ColorPicker
+ getBackgroundColorPicker() : ColorPicker
+ getCanvas() : Pane
+ initLayout() : void
+ initControllers() : void
+ initStyle() : void
+ reloadWorkspace(data : AppDataComponent) : void
mmmStyle
MetroLine
Station
mmmFiles
VBox
FlowPane
HBox
Label
Button
ColorPicker
Slider
mapEditController
app : AppTemplate
dataManager : mmmData
+ mapEditController(initApp : AppTemplate)
+ processAboutRequest() : void
+ processUndoRequest() : void
+ processRedoRequest() : void
+ processAddLineRequest() : void
+ processRemoveLineRequest() : void
+ processAddStationRequest() : void
+ processRemoveStationRequest() : void
+ processListLinesRequest() : void
+ processThicknessSliderPress() : void
+ processThicknessSliderRelease() : void
+ processLineThickness() : void
+ processLineColorRequest() : void
+ processAddStationRequest() : void
+ processRemoveStationRequest() : void
+ processSnapRequest() : void
+ processMoveLabelRequest() : void
+ processRotateRequest() : void
+ processStationColorRequest() : void
+ processFindRouteRequest() : void
+ processSetImgBackgroundRequest() : void
+ processSetBackgroundColorRequest() : void
+ processAddImageRequest() : void
+ processAddLabelRequest() : void
+ processRemoveElementRequest() : void
+ processBoldRequest() : void
+ processItalicsRequest() : void
+ processFontSizeRequest() : void
+ processFontStyleRequest() : void
+ processFontColorRequest() : void
+ processZoomInRequest() : void
+ processZoomOutRequest() : void
+ processIncreaseMapRequest() : void
+ processDecreaseMapRequest() : void
CanvasController
app : AppTemplate
startedDragging : boolean
startX : int
startY : int
startMouseX : int
startMouseY : int
+ CanvasController(initApp : AppTemplate)
+ processCanvasMousePress(x : int, y : int) : void
+ processCanvasMouseDragged(x : int, y : int) : void
+ processCanvasMouseRelease(x : int, y : int) : void
mmmData
djf
AppTemplate
AppGUI

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

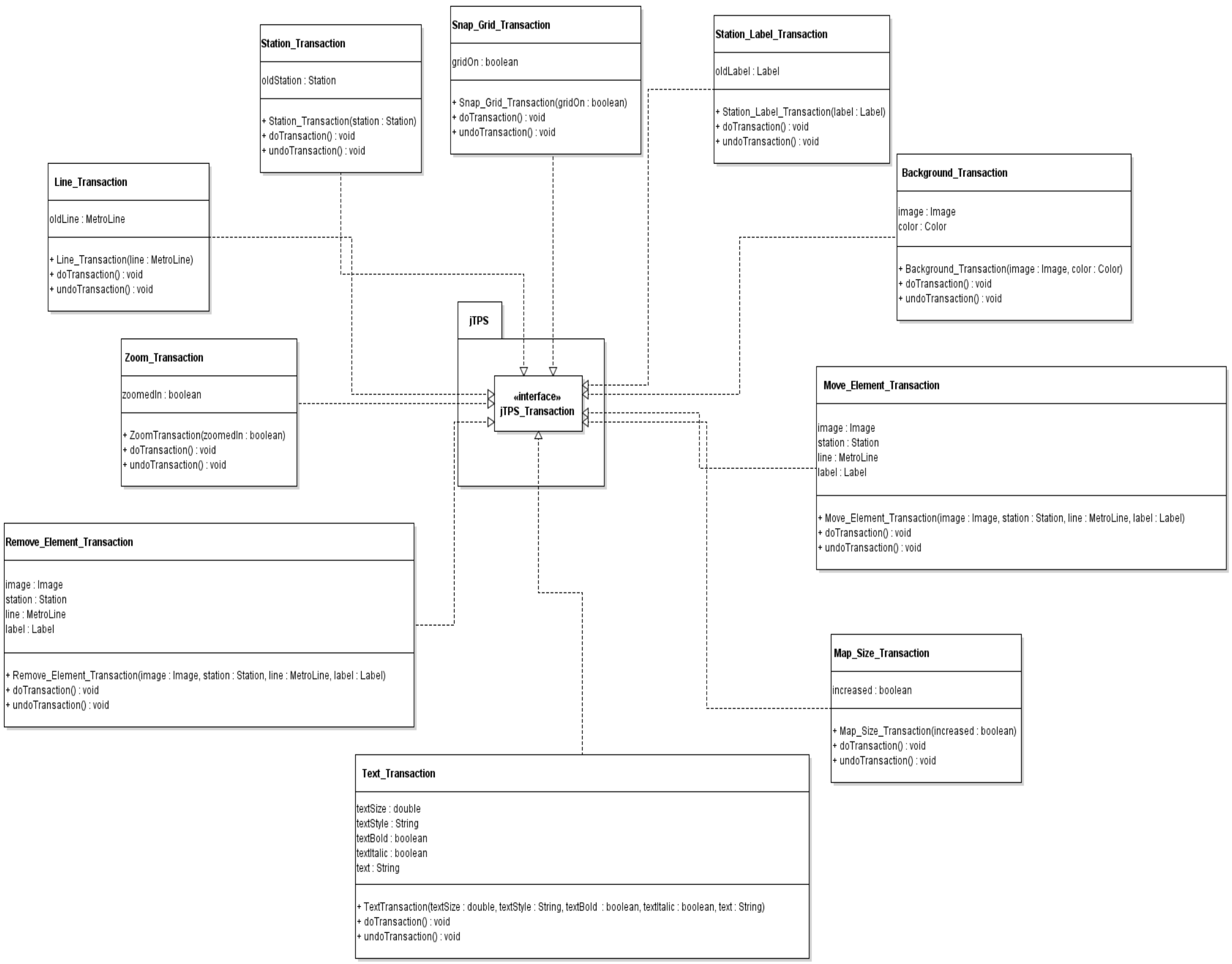

Station_Transaction
oldStation : Station
+ Station_Transaction(station : Station)
+ doTransaction() : void
+ undoTransaction() : void
Snap_Grid_Transaction
gridOn : boolean
+ Snap_Grid_Transaction(gridOn : boolean)
+ doTransaction() : void
+ undoTransaction() : void
Station_Label_Transaction
oldLabel : Label
+ Station_Label_Transaction(label : Label)
+ doTransaction() : void
+ undoTransaction() : void
Line_Transaction
oldLine : MetroLine
+ Line_Transaction(line : MetroLine)
+ doTransaction() : void
+ undoTransaction() : void
Background_Transaction
image : Image
color : Color
+ Background_Transaction(image : Image, color : Color)
+ doTransaction() : void
+ undoTransaction() : void
jTPS
«interface»
jTPS_Transaction
Zoom_Transaction
zoomedIn : boolean
+ ZoomTransaction(zoomedIn : boolean)
+ doTransaction() : void
+ undoTransaction() : void
Move_Element_Transaction
image : Image
station : Station
line : MetroLine
label : Label
+ Move_Element_Transaction(image : Image, station : Station, line : MetroLine, label : Label)
+ doTransaction() : void
+ undoTransaction() : void
Remove_Element_Transaction
image : Image
station : Station
line : MetroLine
label : Label
+ Remove_Element_Transaction(image : Image, station : Station, line : MetroLine, label : Label)
+ doTransaction() : void
+ undoTransaction() : void
Map_Size_Transaction
increased : boolean
+ Map_Size_Transaction(increased : boolean)
+ doTransaction() : void
+ undoTransaction() : void
Text_Transaction
textSize : double
textStyle : String
textBold : boolean
textItalic : boolean
text : String
+ TextTransaction(textSize : double, textStyle : String, textBold : boolean, textItalic : boolean, text : String)
+ doTransaction() : void
+ undoTransaction() : void

Figure 3.8: Detailed Transactions UML Class Diagram

6 Supporting Information

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

6.1: Table of contents

6.2: Appendixes

N/A