The *Metro Map Maker* ™
Software Design Description

Author: Dhruv Sheth
October 2017
Version 1.0

Abstract: This document describes the software design for the Metro Map Maker, a tool to make representations of city subway systems.

Based on IEEE Std 1016™-2009 document format

1 Introduction

Finding your way around a new city can be challenging so many people look to the Internet for help. Cities with subway systems typically provide maps to help one navigate from one stop to another across a number of intersecting lines. These maps let one chart which lines to take and how many stops it will be before one arrives and the user can typically choose between multiple routes.

The ***Metro Map Maker (i.e. M^3)*** application will provide 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. Finally, it will provide an export feature such that it may export a generated map and associated metro system information to a format that can be used by a corresponding Web application that will be able to make use of it.

1.1 **Purpose**

The purpose of this document is to specify how our ***Metro Map Maker*** program should look and operate. The intended audience for this document is all the members of the development team, those who will design the maps for use with the Web application, and the potential users of such an application. This document serves as an agreement among all parties and as a reference for how the map creation tool should ultimately be constructed. Upon completing the reading of this document, one should clearly visualize how the application will look and operate.

1.2 Scope

For this project the goal is for users to easily make and edit subway maps. There will be an emphasis on ease of use. Note that 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.

1.3 Definitions, acronyms, and abbreviations

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.

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

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

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.

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.

1.4 References

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

1.5 Overview

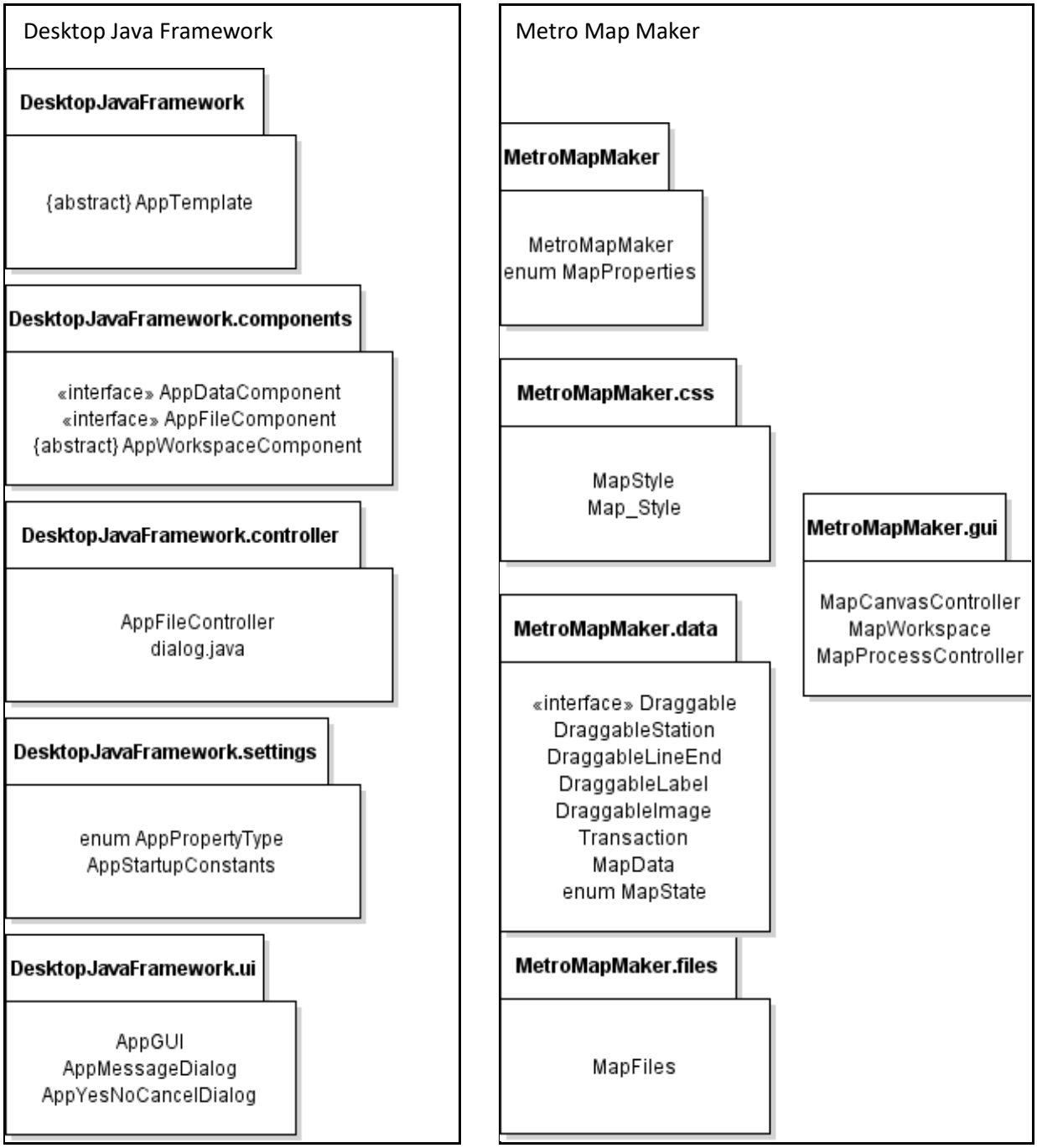
This Software Design Description document provides a working design for the ***Metro Map Maker*** software application as described in the ***Metro Map Maker*** Software Requirements Specification. Note that all parties in the implementation stage must agree upon all connections between components before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

2 Package-Level Design Viewpoint

As mentioned, this design will encompass The ***Metro Map Maker*** application, the **Desktop Java Framework** and jTPS which are to be used in ***Metro Map Maker***'s construction. In building both 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.

2.1 Metro Map Maker and Desktop Java Framework overview

The Metro Map Maker, Desktop Java Framework and jTPS will be designed and developed in tandem. Figure 2.1 specifies all the components to be developed and places all classes in home packages.



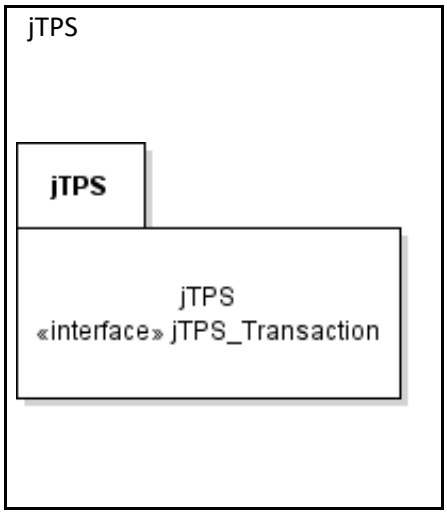


Figure 2.1: Design Packages Overview

2.2 Java API Usage

Both the Desktop Java framework and the Metro Map Maker will be developed using the Java programming languages.

As such, this design will make use of the classes specified in Figure 2.2.

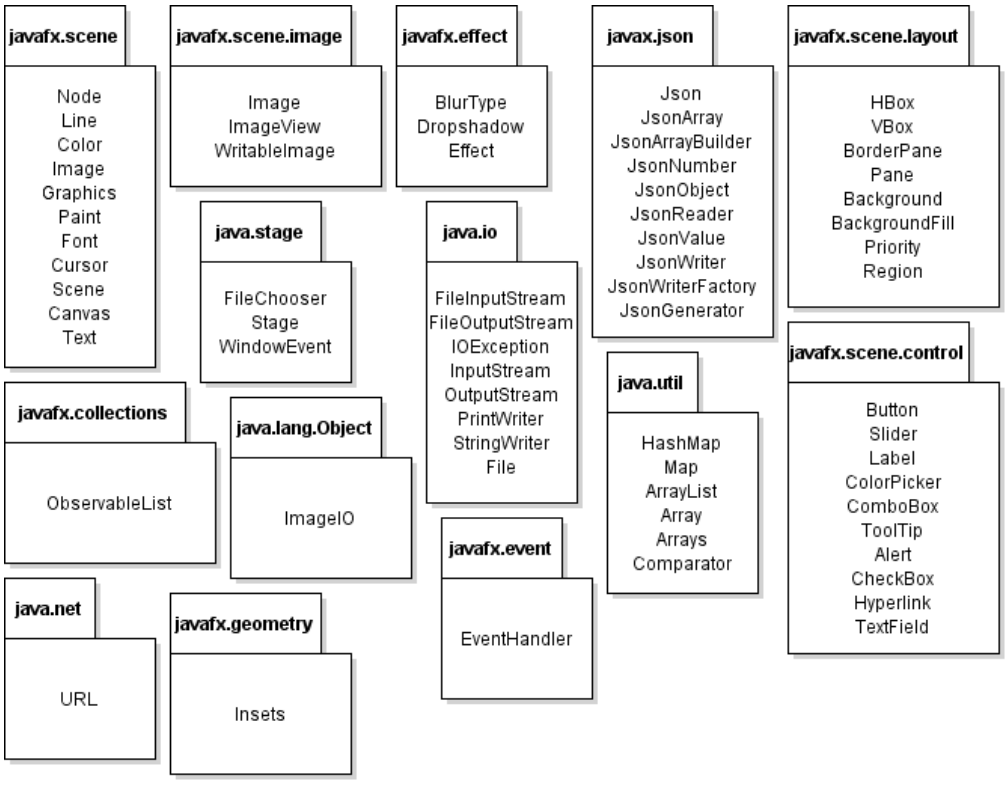


Figure 2.2: Java API Classes and Packages To Be Used

2.3 Java API Usage Descriptions

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

Class/Interface	Use
Color	For setting the rendering colors for text and Metro Lines
Font	For setting the fonts for rendered text
Graphics	For rendering text, images, and shapes to the canvas
Node	For all the text, images, and shapes on the canvas
Line	For Metro Line on canvas
Paint	For default background colors
Cursor	For different modes for cursor
Scene	For setting scene for Stage
Image	For adding Images to the canvas
Canvas	For showing the rendered content
Text	For Station Names

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

Class/Interface	Use
Image	For Image Overlay.
ImageView	For viewing the Images on canvas as ImageView is a subclass of Node
WritableImage	For exporting images from canvas in a png format

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

Class/Interface	Use
BlurType	For highlighting the Node, this will add blue to the effect
DropShadow	For highlighting the Node, this will add shadow at the end of the effect
Effect	For activating that effect on a Node

Table 2.3: Uses for classes in the Java API's javafx.effect package

Class/Interface	Use
Json	For creating Json Objects
JsonArray	For creating an immutable JSON array (an ordered sequence)
JsonArrayBuilder	A builder for creating JsonArray models from scratch
JsonNumber	An immutable JSON number value for storing in JsonArray
JsonObject	For representing an immutable JSON object value
JsonReader	For reading a JSON object or an array structure from input source
JsonValue	For representing an immutable JSON value
JsonWriter	For writing Json object or array structure to an output source
JsonWriterFactory	For creating JsonWriter instances
JsonGenerator	For writing data to an output source in a streaming way

Table 2.4: Uses for classes in the Java API's javax.json package

Class/Interface	Use
HBox	For setting UI for the editToolbar
VBox	For setting UI for the editToolbar
BorderPane	For the main workspace UI
Pane	For the Canvas
Background	For setting the background color of Canvas
BackgroundFill	For creating a fill with color for Background
Priority	For setting priorities(or an order) for some layout classes
Region	For Toolbar to resize on window size change

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

Class/Interface	Use
FileChooser	For making a dialog for selecting files and creating from directory
Stage	For making a window for our application
WindowEvent	For set On Close when welcome dialog is closed

Table 2.6: Uses for classes in the Java API's java.stage package

Class/Interface	Use
FileInputStream	For obtaining input bytes from a file in a file system
FileOutputStream	For creating an output stream for writing data to a File
IOException	For try to catch to catch this exception in case of error
InputStream	For getting the next byte of input.
OutputStream	For writing data to a File
PrintWriter	For printing text data to the File
StringWriter	For collecting output in a string buffer
File	For creating a new file in the directory

Table 2.7: Uses for classes in the Java API's java.io package

Class/Interface	Use
HashMap	For JsonGenerator to make write the Objects into the files
Map	For JsonGenerator to make write the Objects into the files
ArrayList	For making temporary list of Nodes
Array	For going through Files in a directory to show recent files in Welcome Dialog
Arrays	For converting Array to ArrayList
Comparator	For sorting the files according to date modified

Table 2.8: Uses for classes in the Java API's java.util package

Class/Interface	Use
ObservableList	For working with Pane children (Add , Remove and Edit Nodes)

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

Class/Interface	Use
ImageIO	For saving a snapshot of canvas into an image

Table 2.10: Uses for classes in the Java API's java.lang.Object package

Class/Interface	Use
Button	For making Buttons for UI
Slider	For Line Thickness and Station CircleRadius
Label	For creating labels for Buttons and other UI
ColorPicker	For all the color selection in the application
ComboBox	For selection stations and lines on canvas
Tooltip	For creating tips for all editing options
Alert	For making alerts to get input from user and alert user about an error
CheckBox	For Show Grids to enable and disable
Hyperlink	For showing clickable Text in Welcome Dialog
TextField	For getting the line name and other things from user

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

Class/Interface	Use
URL	For stylesheet URL

Table 2.12: Uses for classes in the Java API's java.net package

Class/Interface	Use
Insets	For setting the size of various Panes

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

Class/Interface	Use
EventHandler	For window event set on close request

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

3 Class-Level Design Viewpoint

As mentioned, this design will encompass both the Metro Map Maker application and the Desktop Java Framework. The following UML Class Diagrams reflect this. 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.

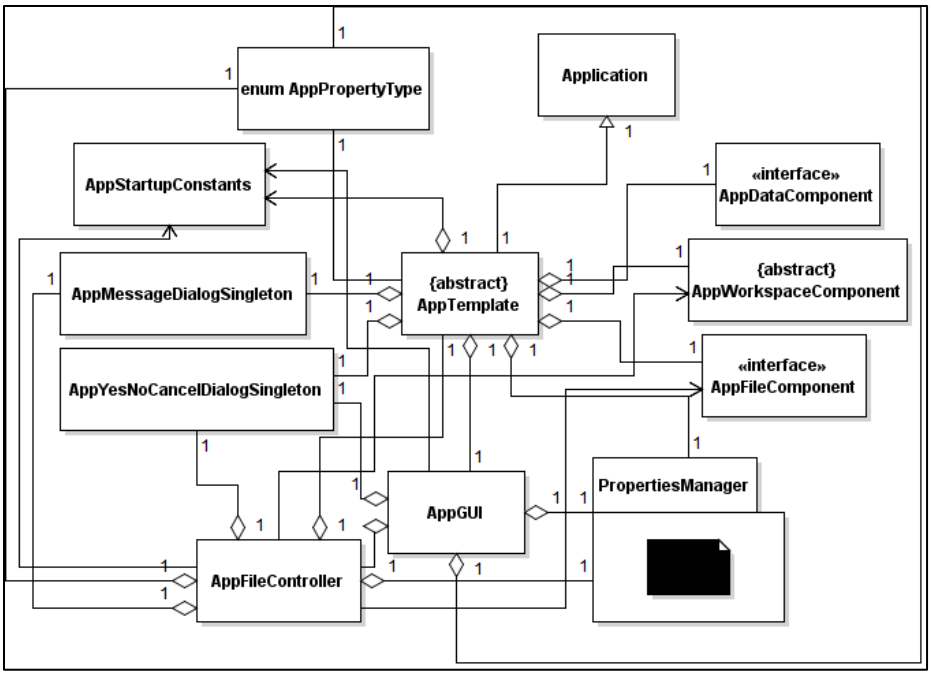


Figure 3.1: Desktop Java Framework Overview UML Class Diagram

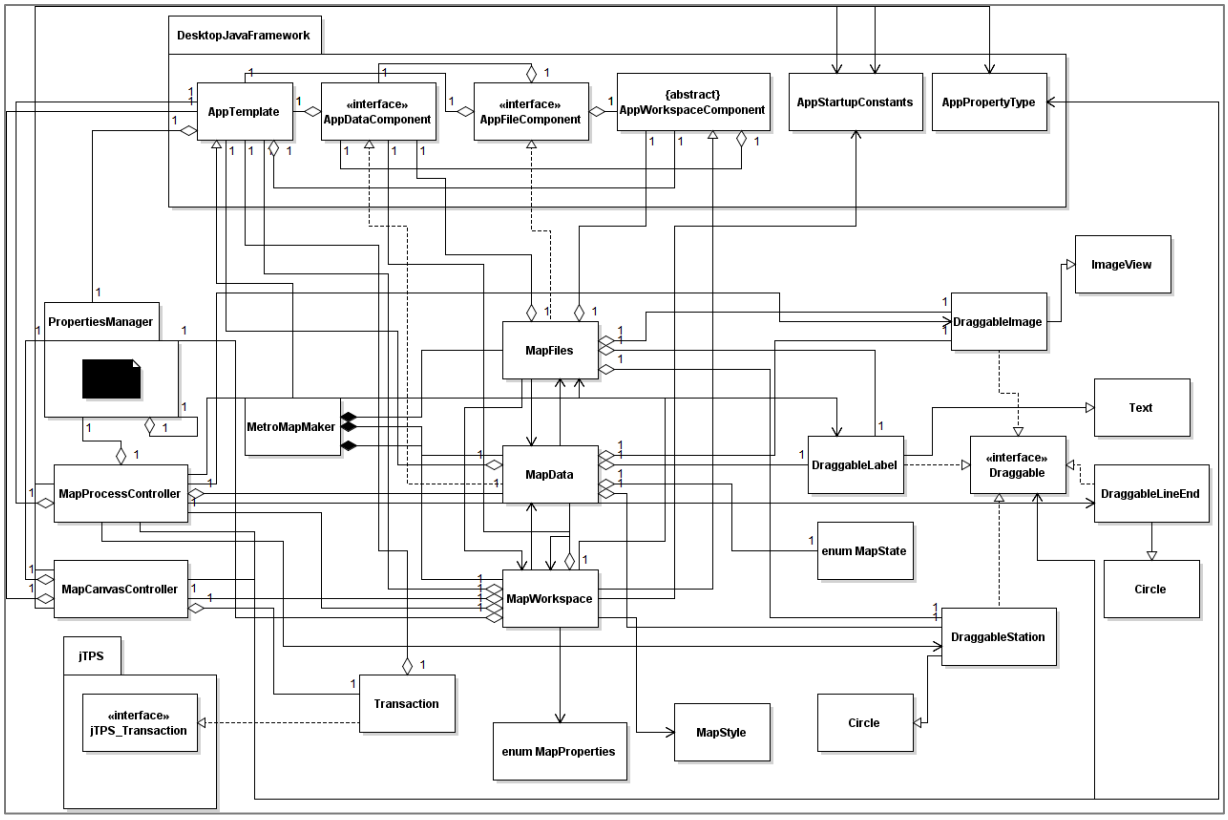


Figure 3.2: Metro Map Maker Overview UML Class Diagram

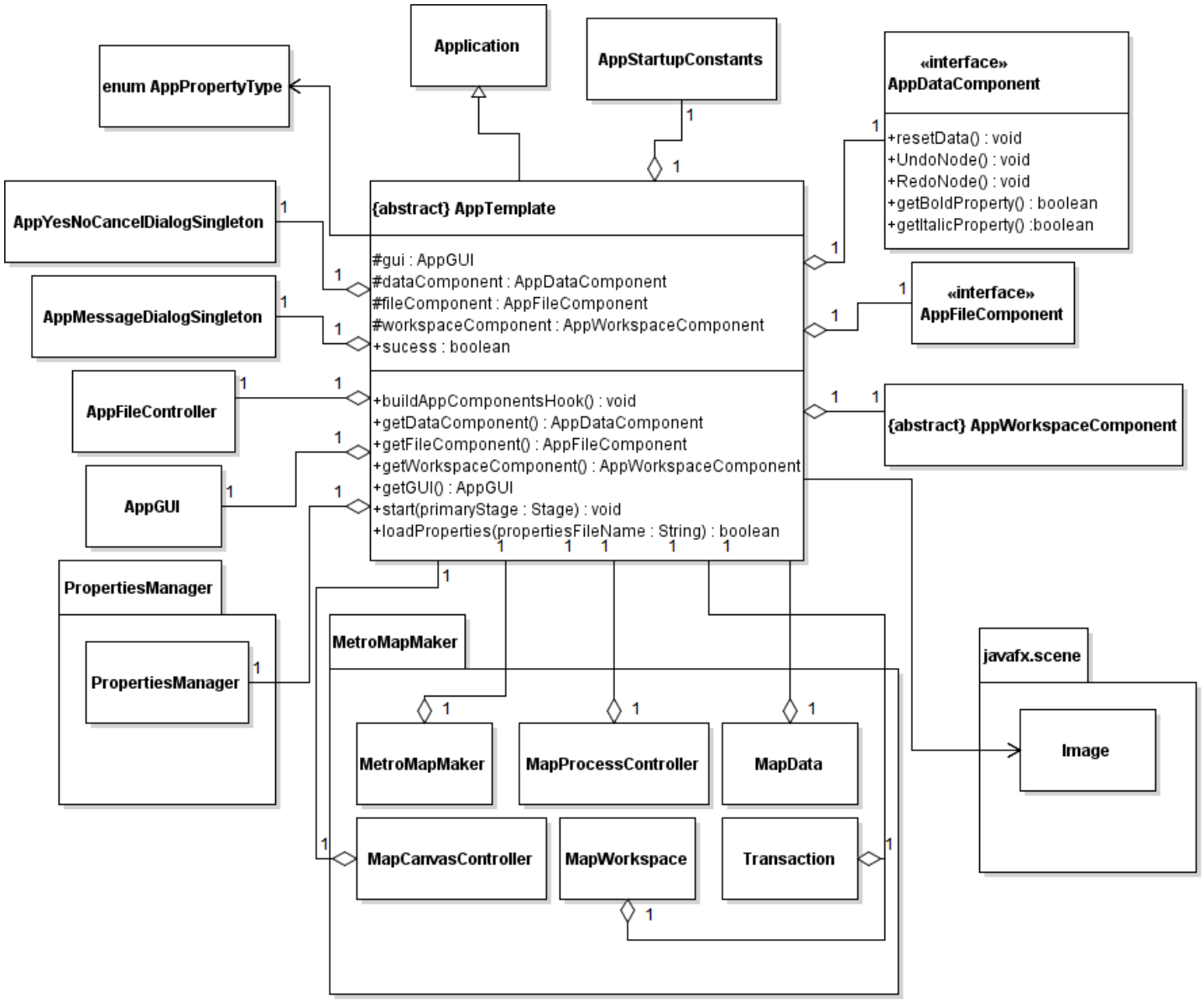


Figure 3.3: Detailed AppTemplate and App Components UML Class Diagram

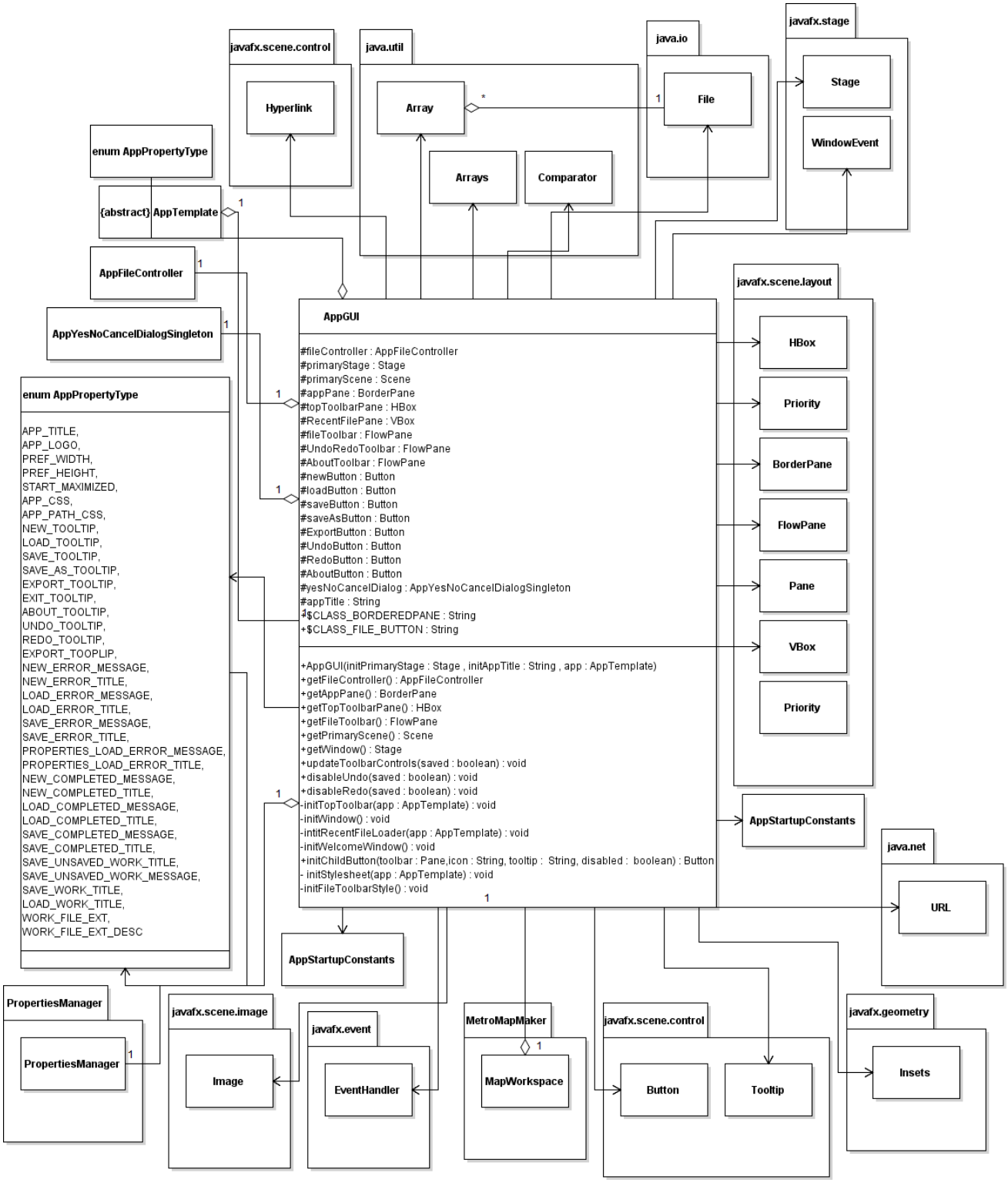


Figure 3.4: Detailed AppGUI and enum AppPropertyType UML Class Diagram

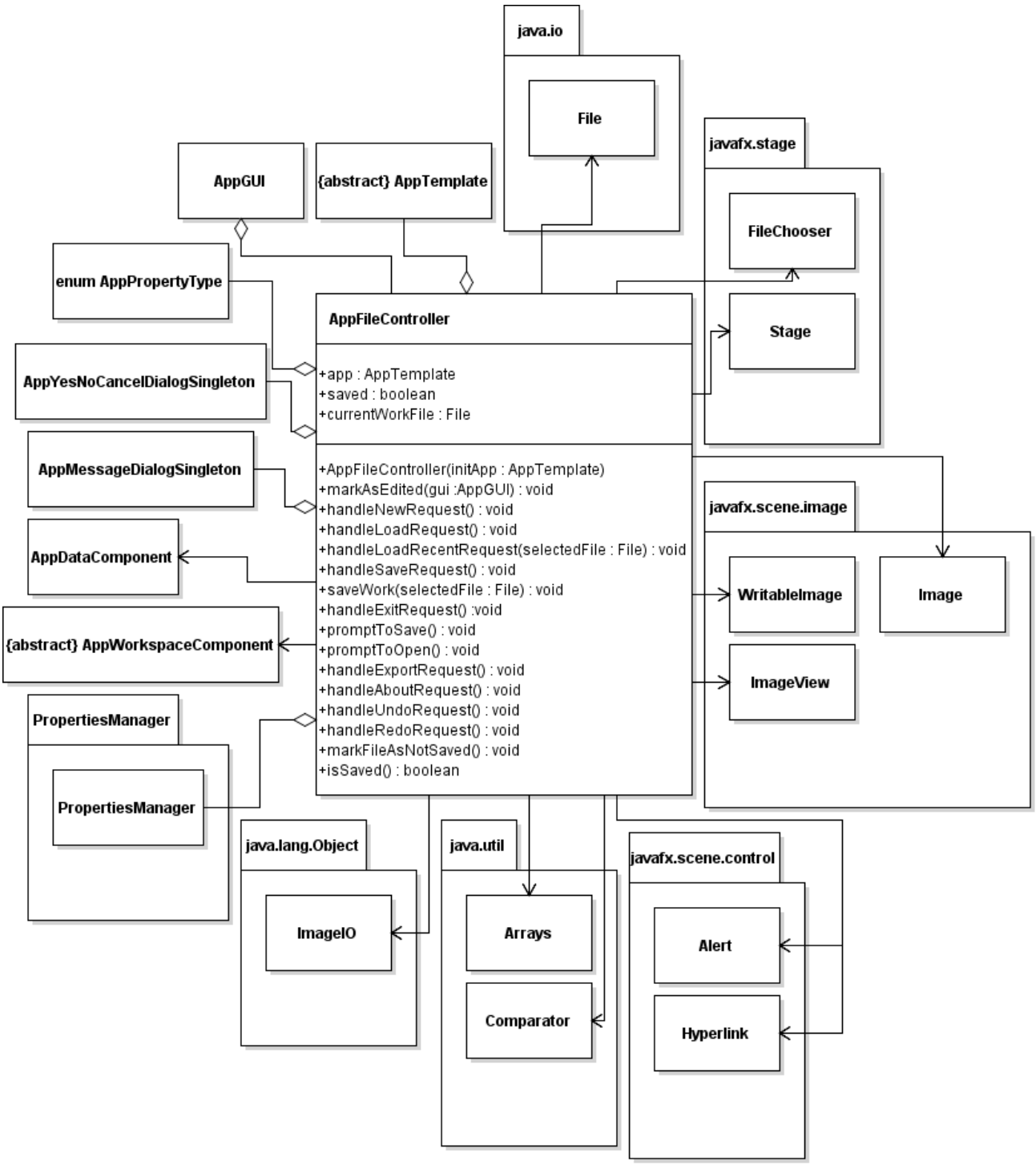


Figure 3.4: Detailed AppFileController UML Class Diagram

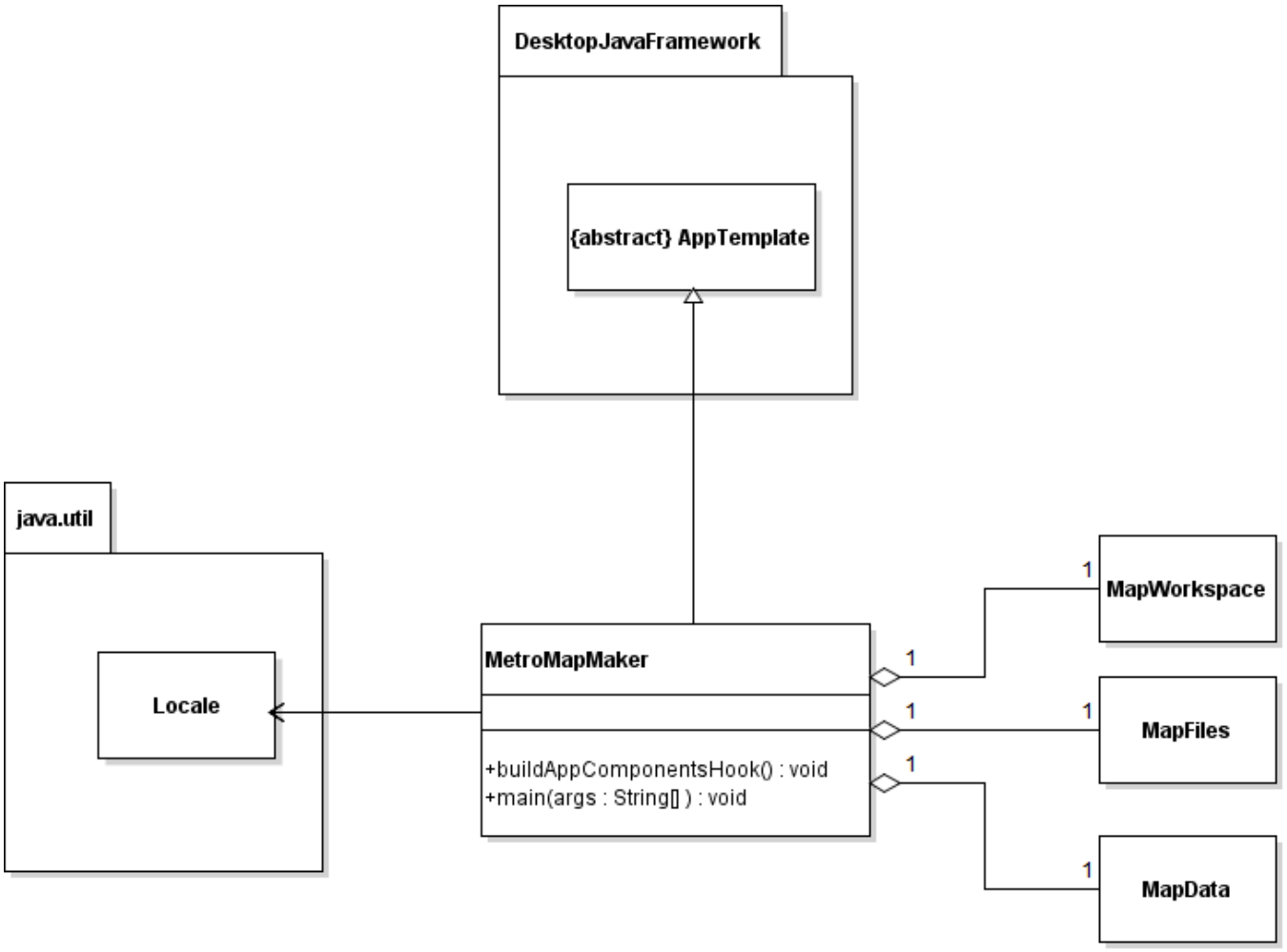


Figure 3.5: Detailed MetroMapMaker UML Class Diagram

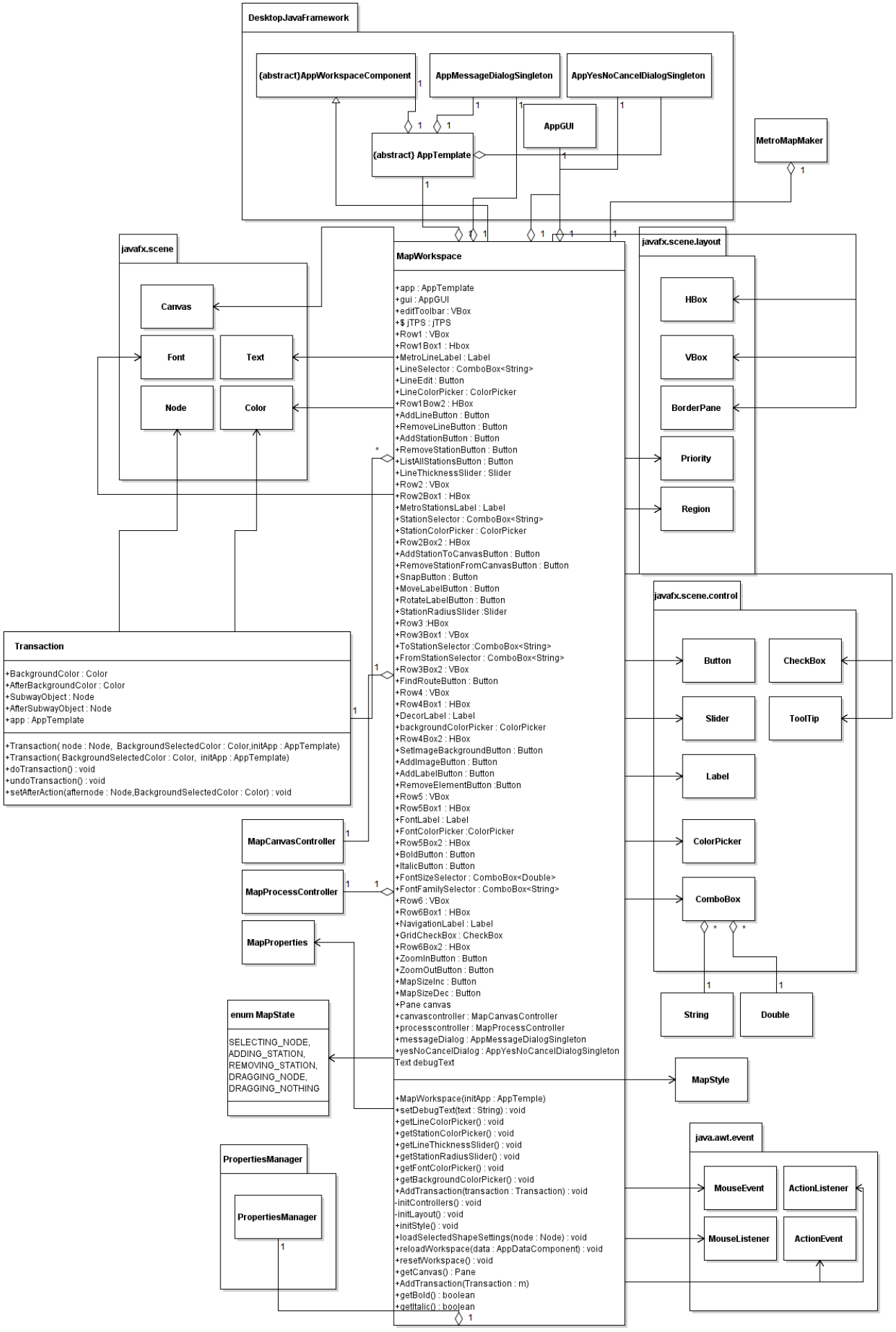


Figure 3.6: Detailed MapWorkspace, MapState and Transaction UML Class Diagrams

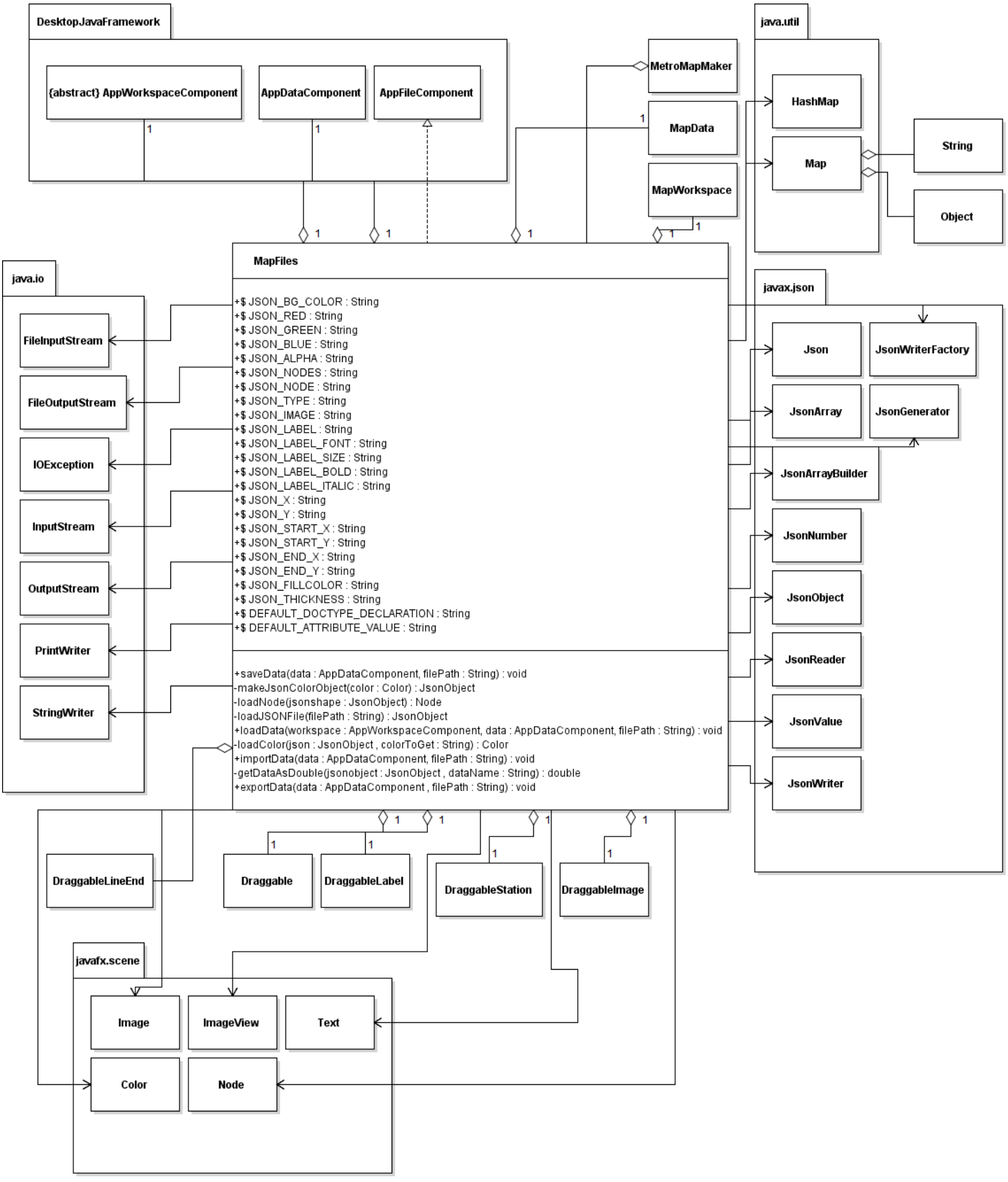


Figure 3.7: Detailed MapFiles UML Class Diagram

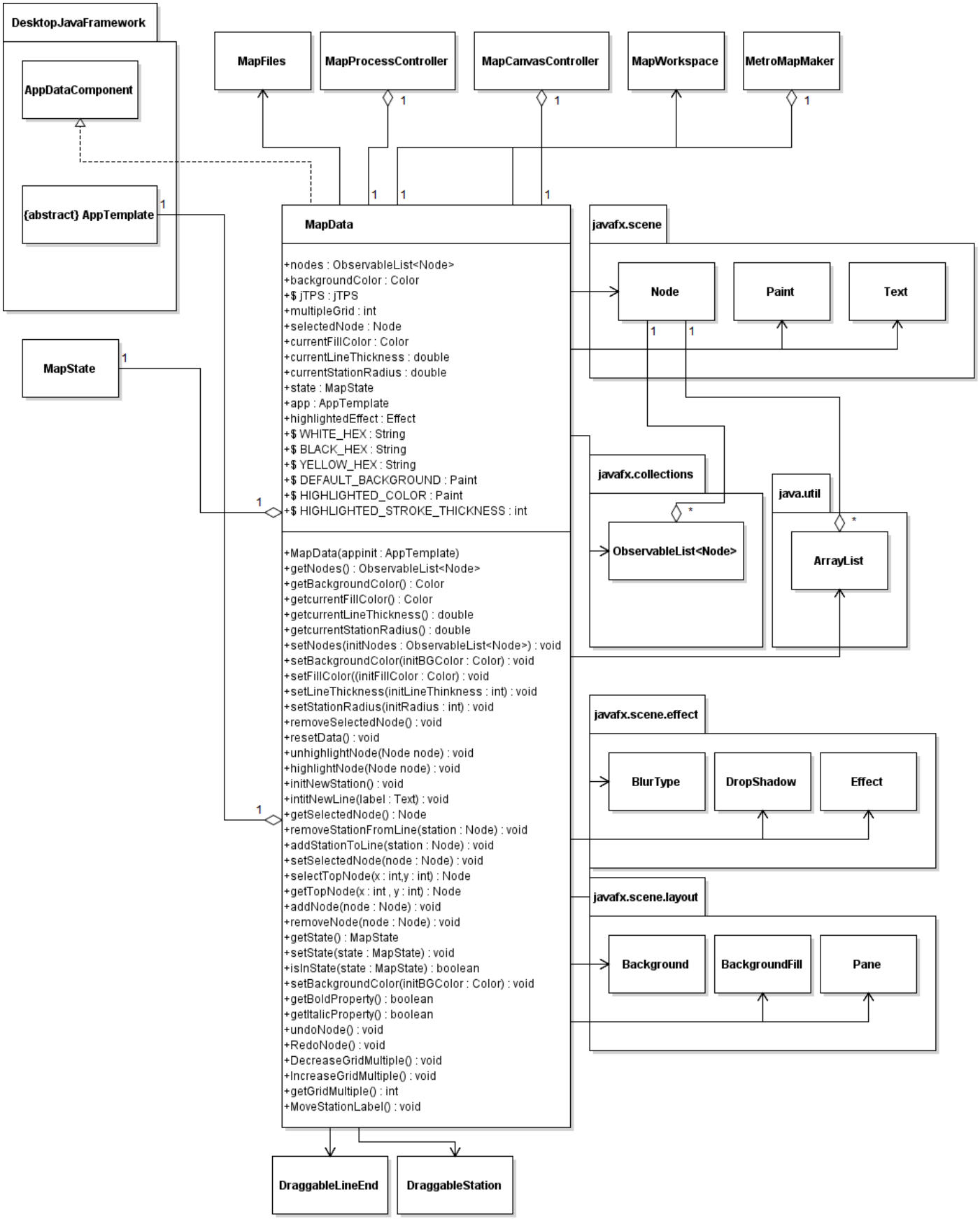


Figure 3.8: Detailed MapData UML Class Diagram

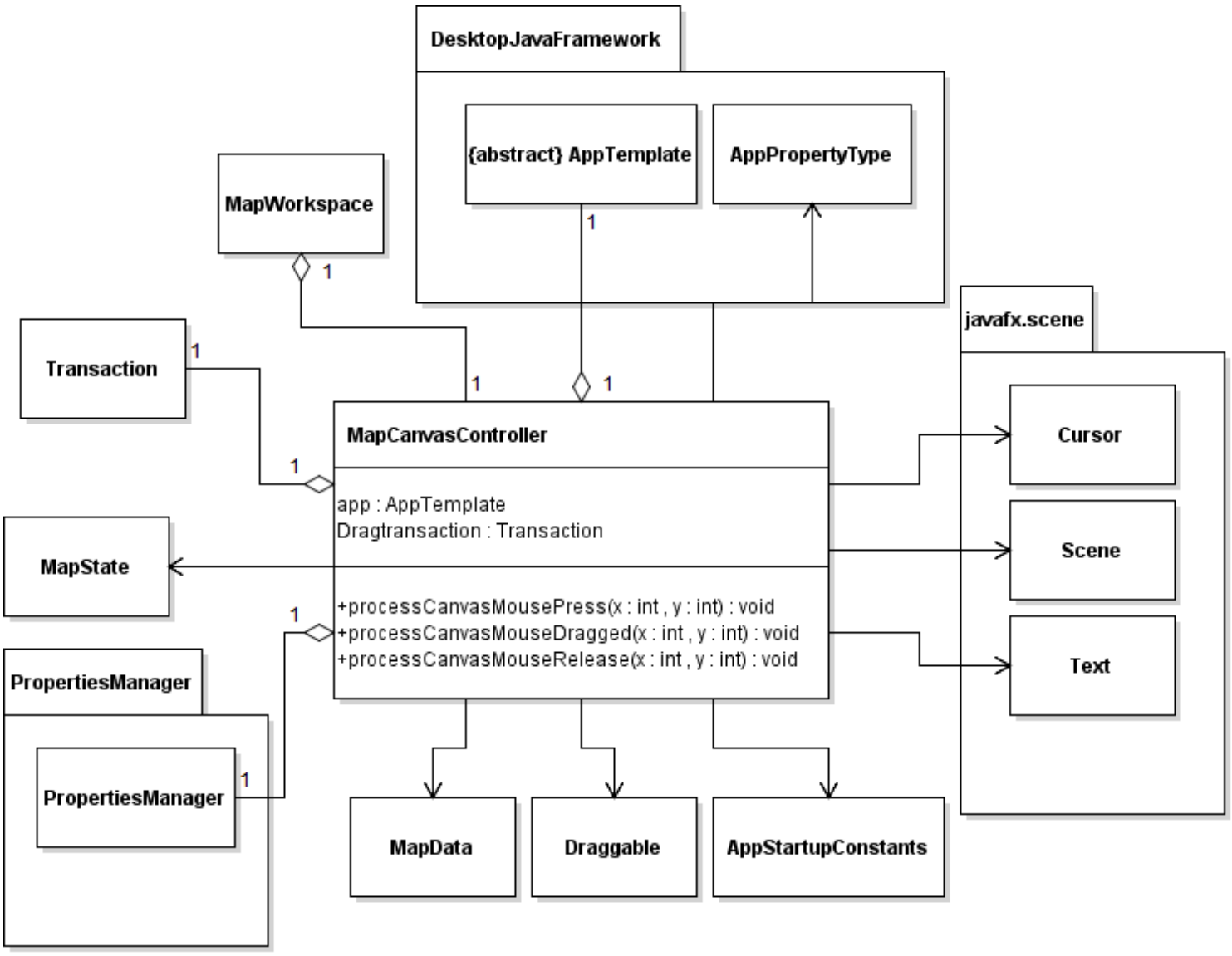


Figure 3.9: Detailed MapCanvasController UML Class Diagram

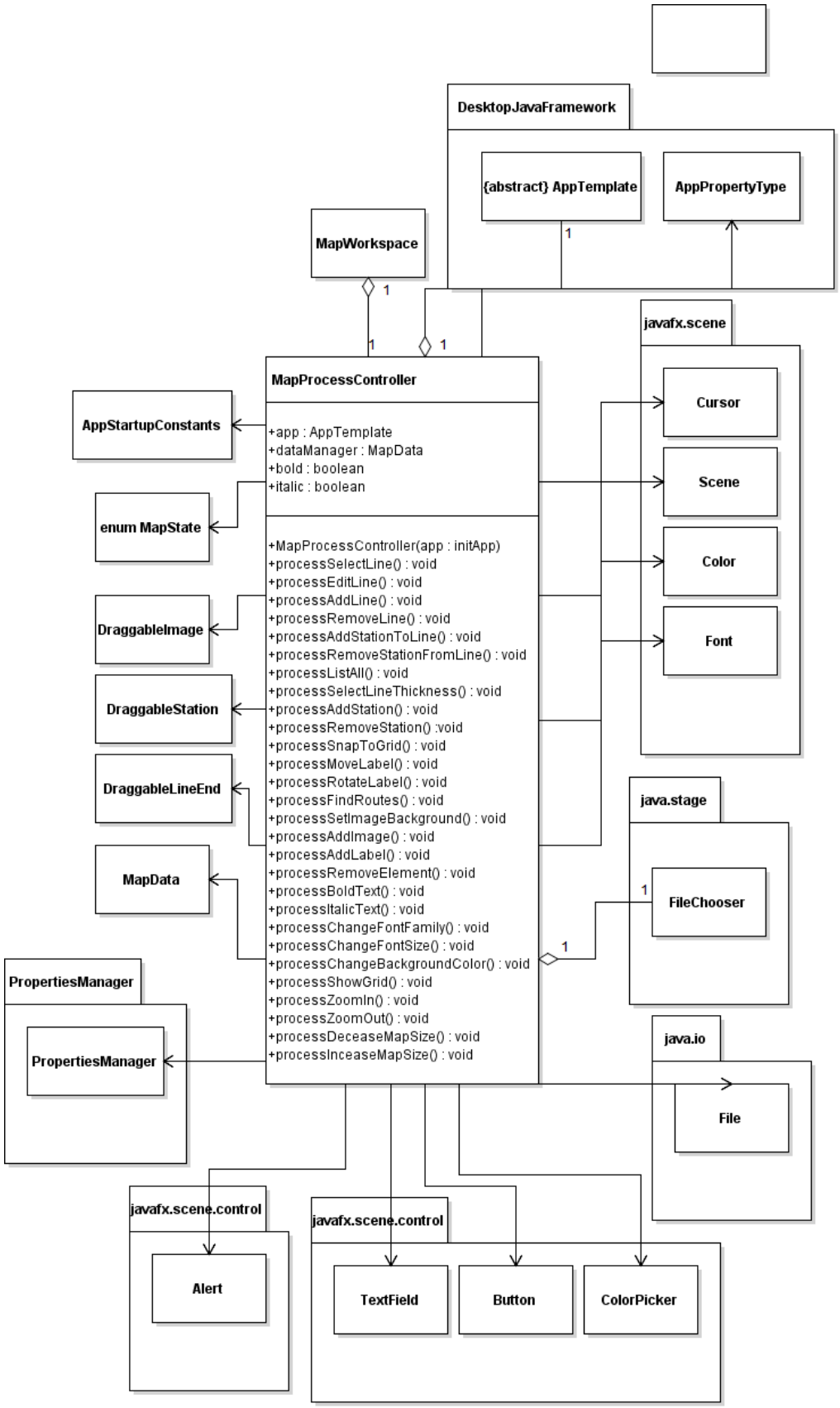


Figure 3.10: Detailed MapProcessController UML Class Diagram

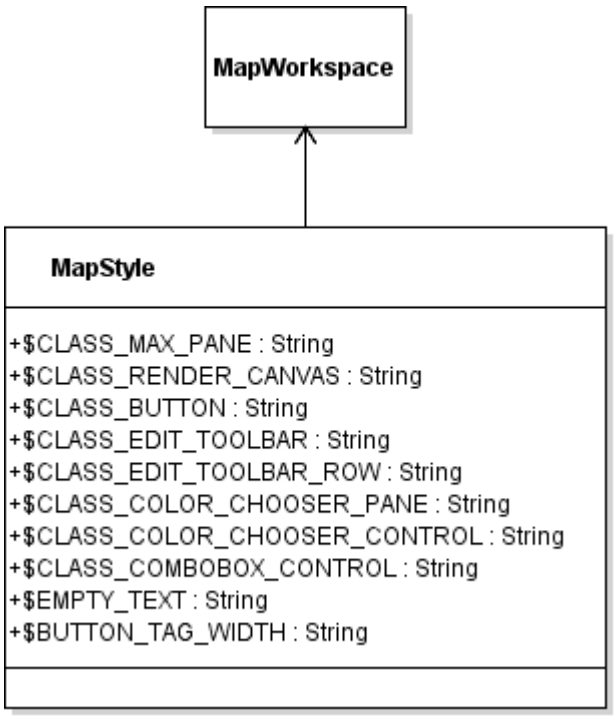


Figure 3.10: Detailed MapStyle UML Class Diagram

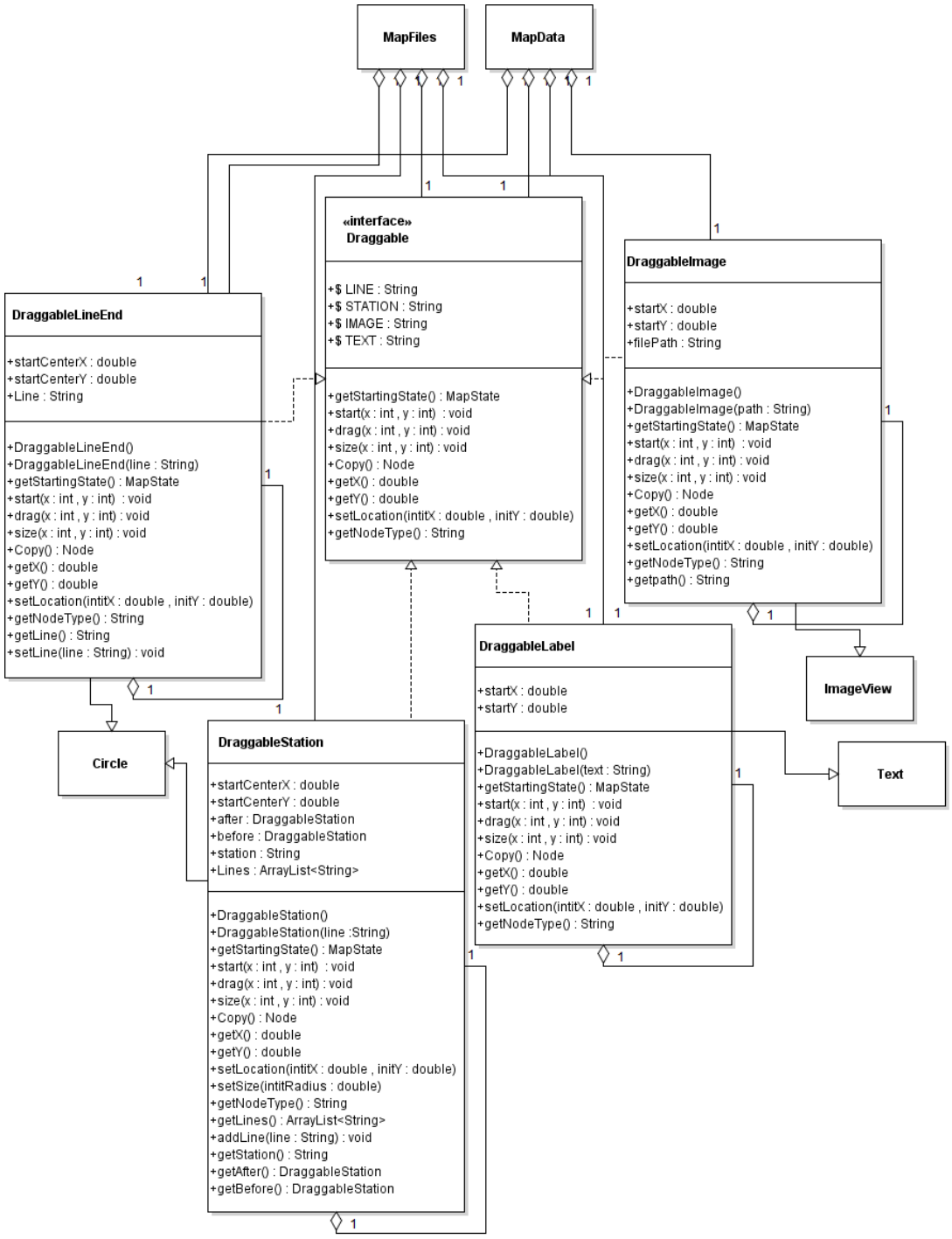


Figure 3.11: Detailed Draggable interface and other Draggable UML Class Diagrams

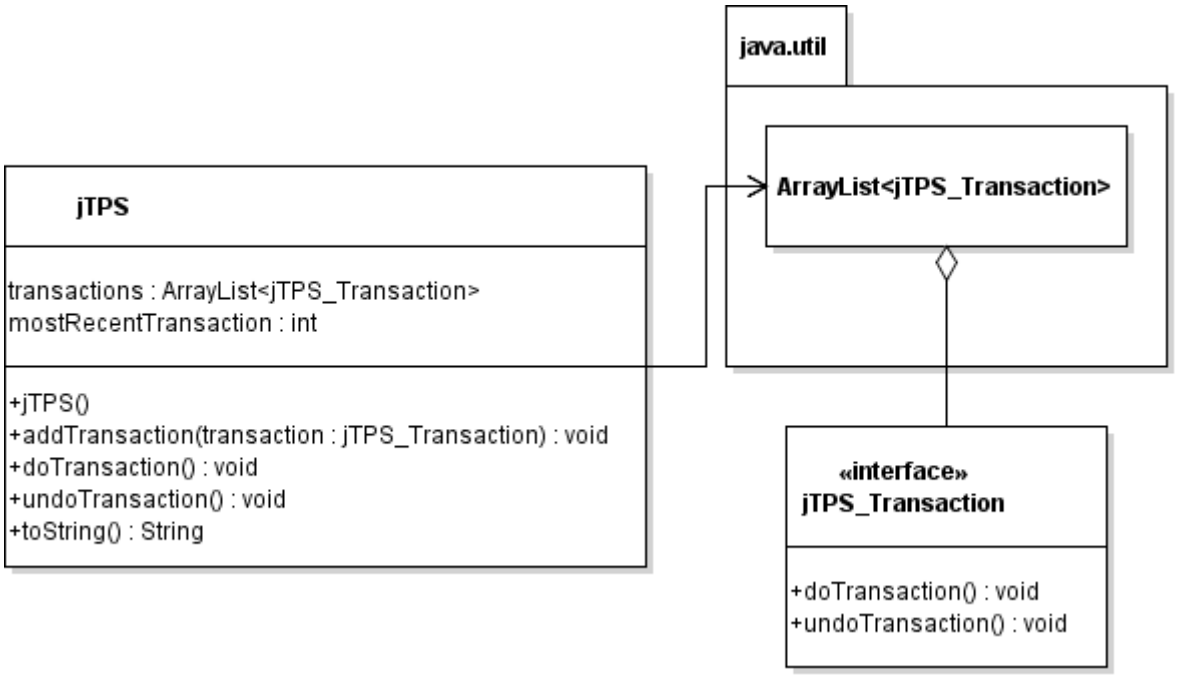
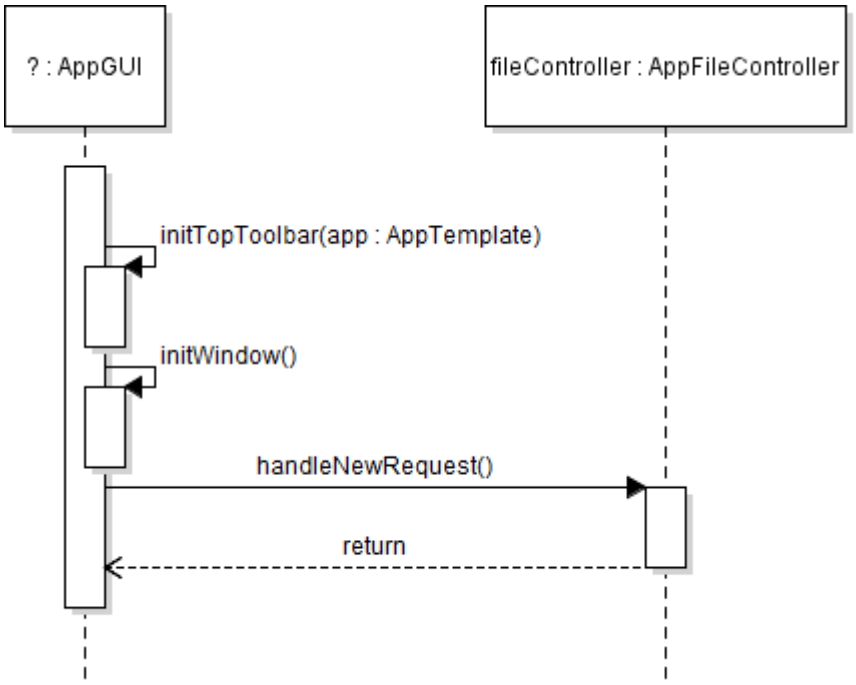
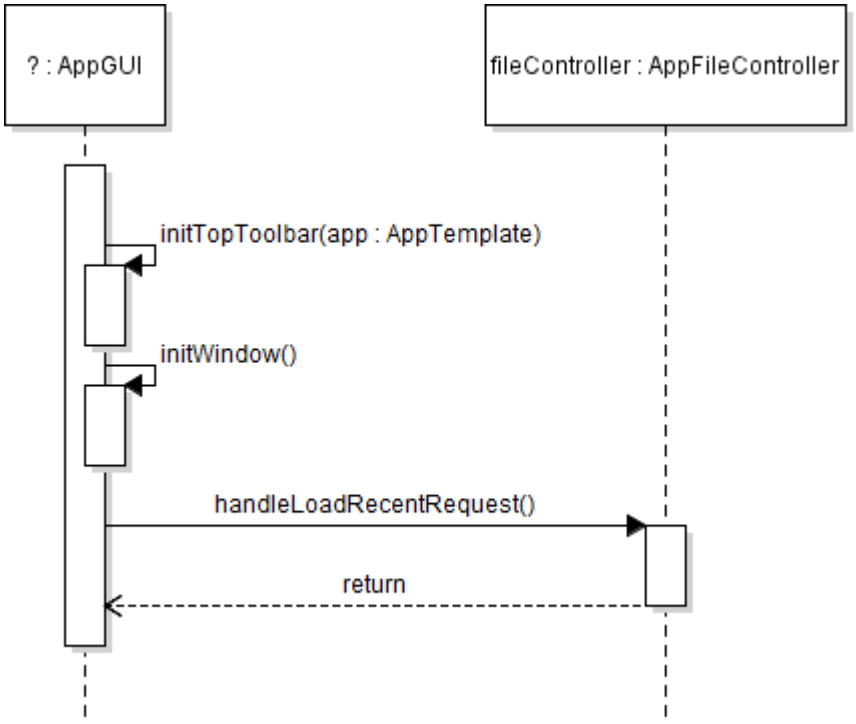


Figure 3.11: Detailed jTPS and interface jTPS_Transaction UML Class Diagrams

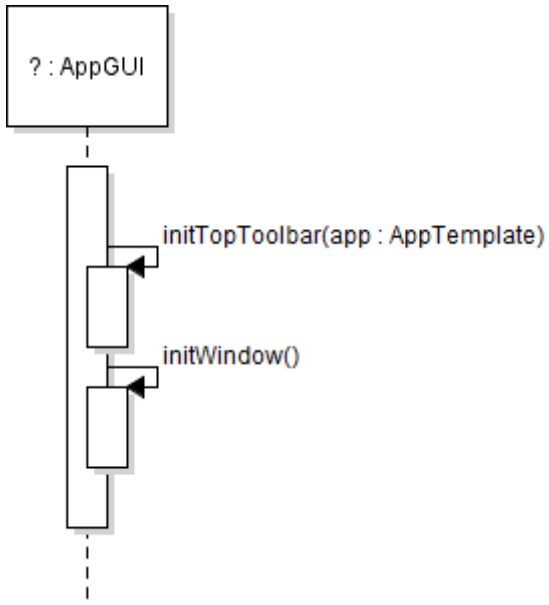
4 Method-Level Design Viewpoint



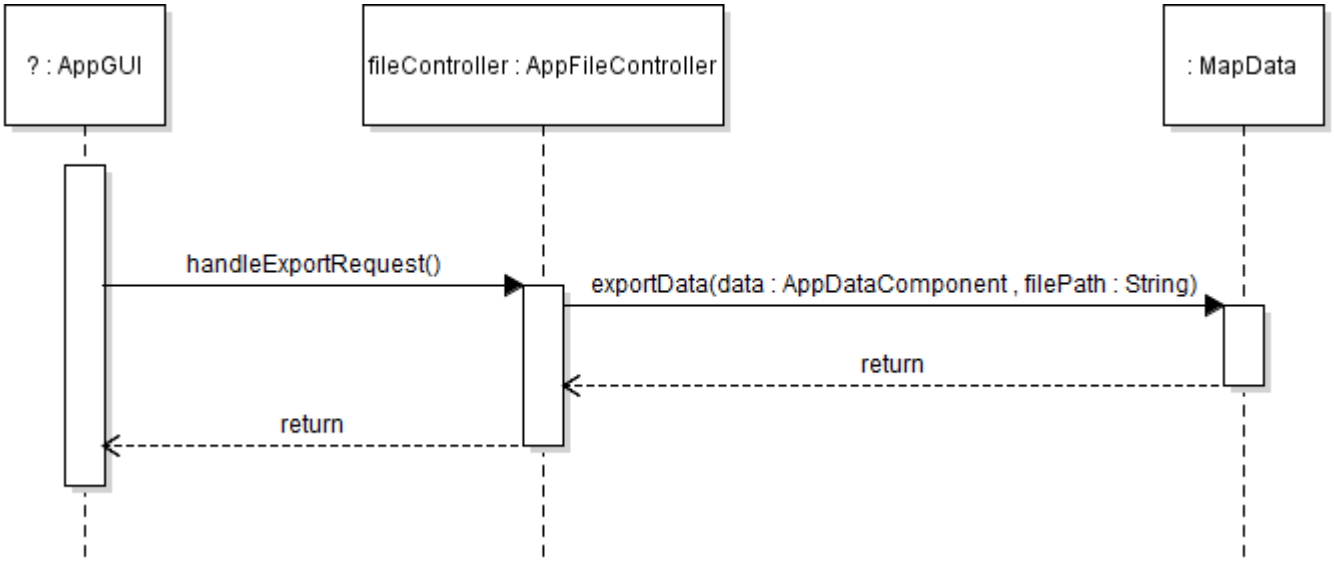
Case 2.1 : Create a New Map Handling



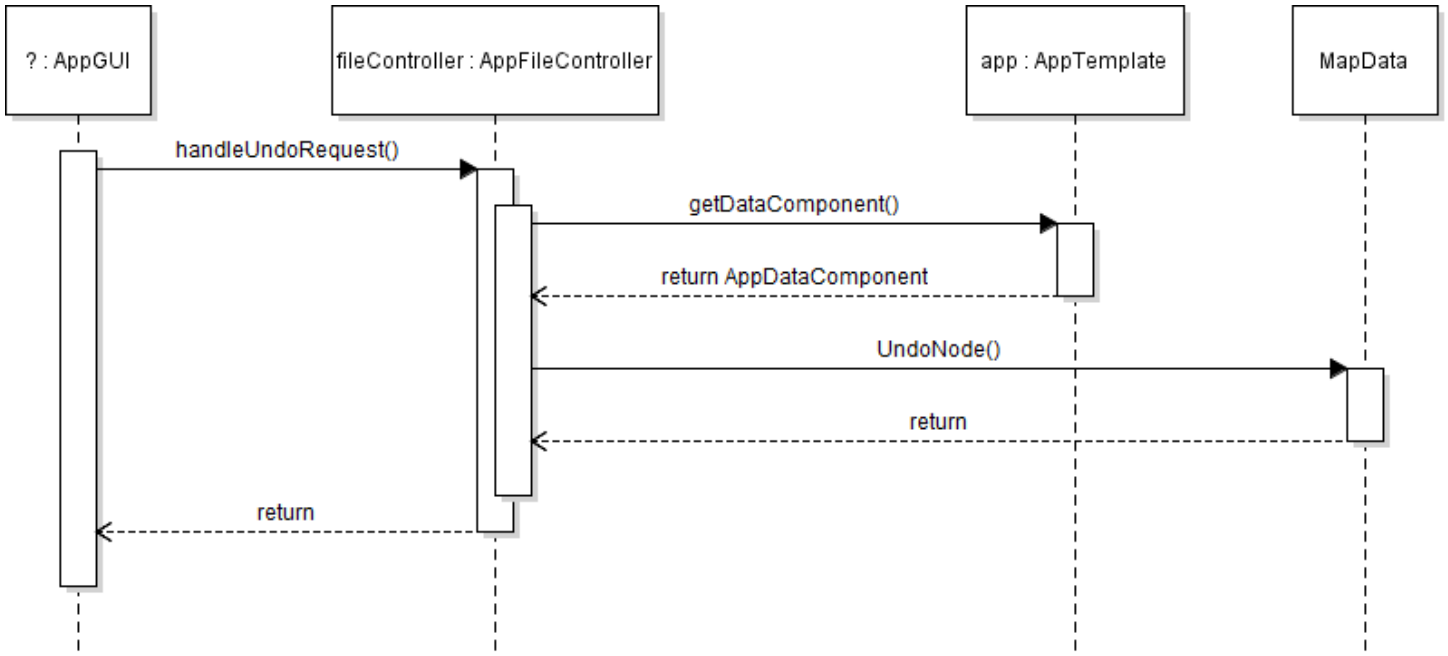
Case 2.2 : Load Recent Map Handling



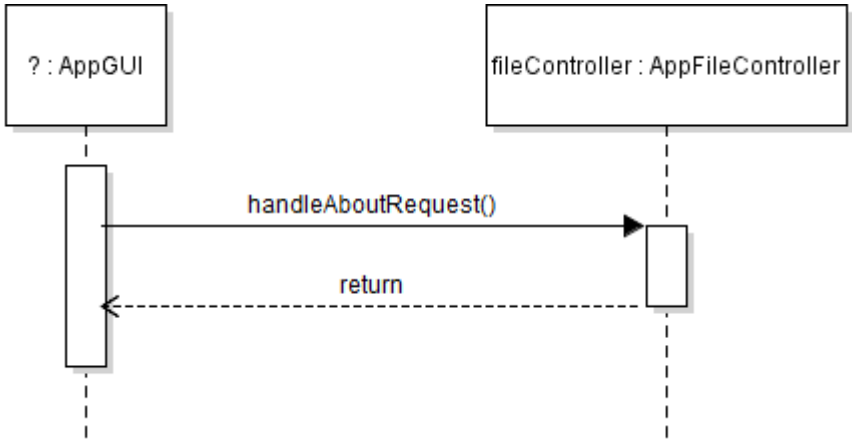
Case 2.3 : Load Recent Map Handling



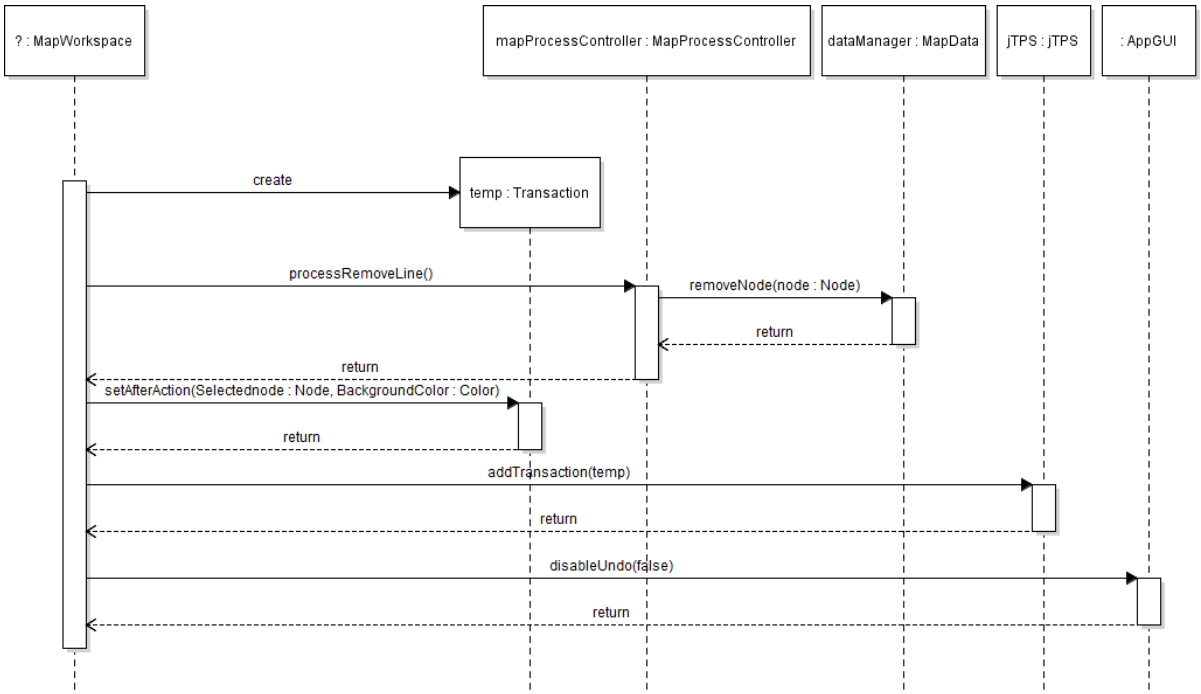
Case 2.8 : Export Map Handling



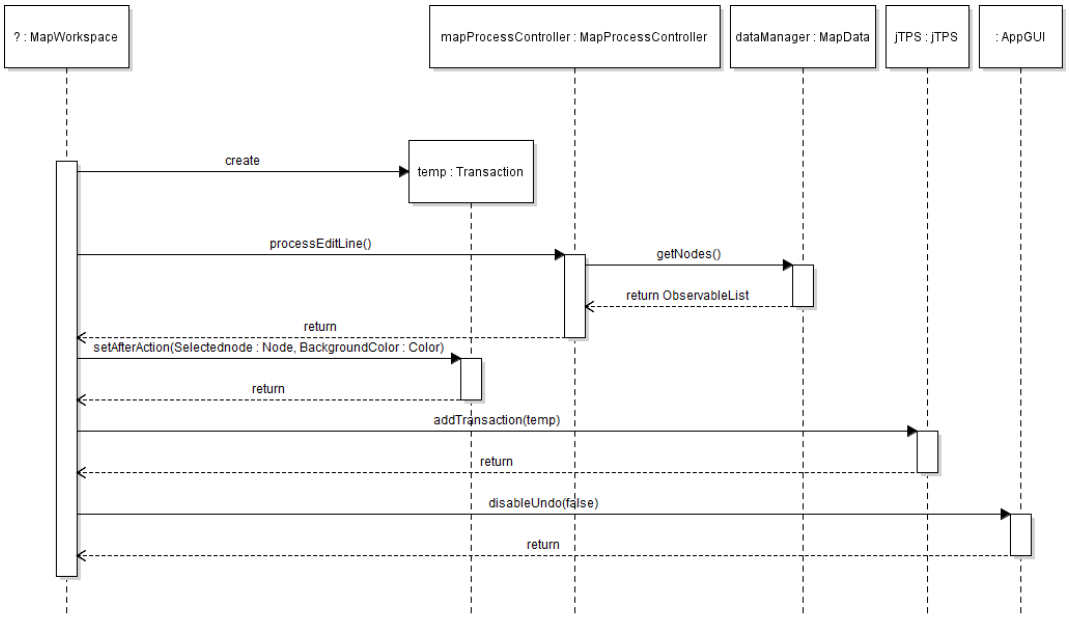
Case 2.9 : Undo Button On Click Action Handling



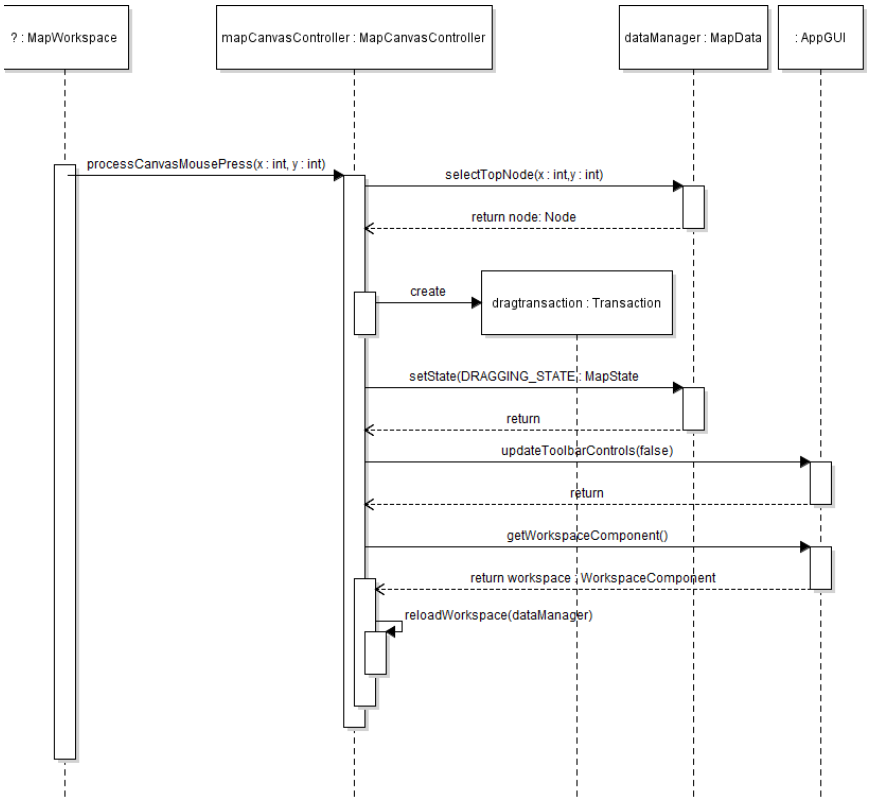
Case 2.11 : About Application Button On Click Action Handling



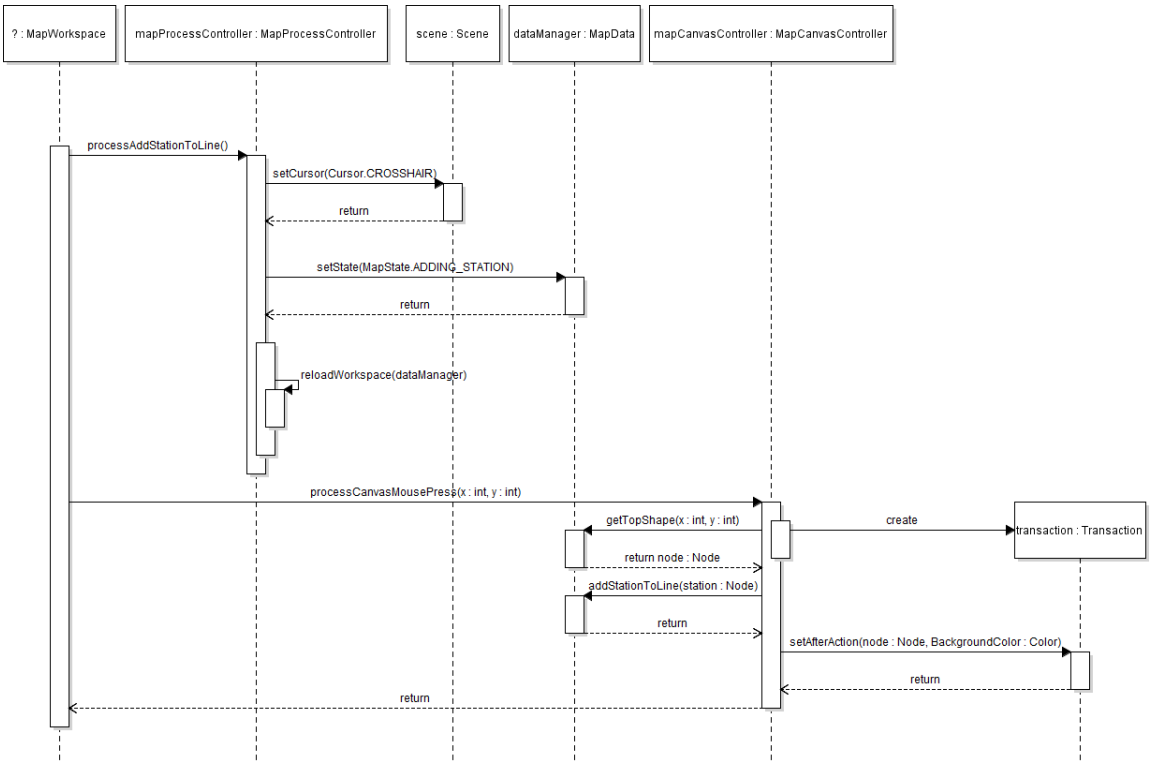
Case 2.13 : Remove Line Handling



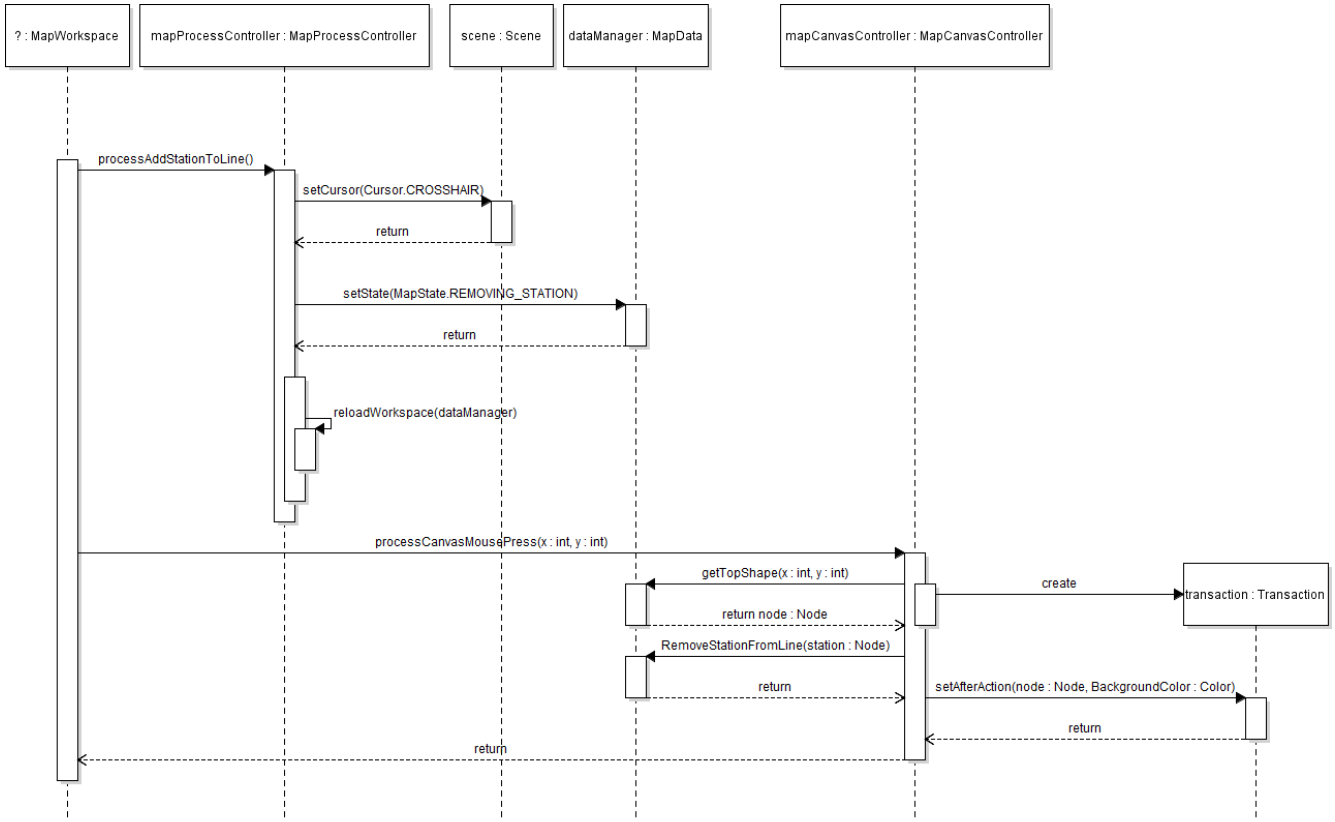
Case 2.14 : Edit Line Button On Click Handling



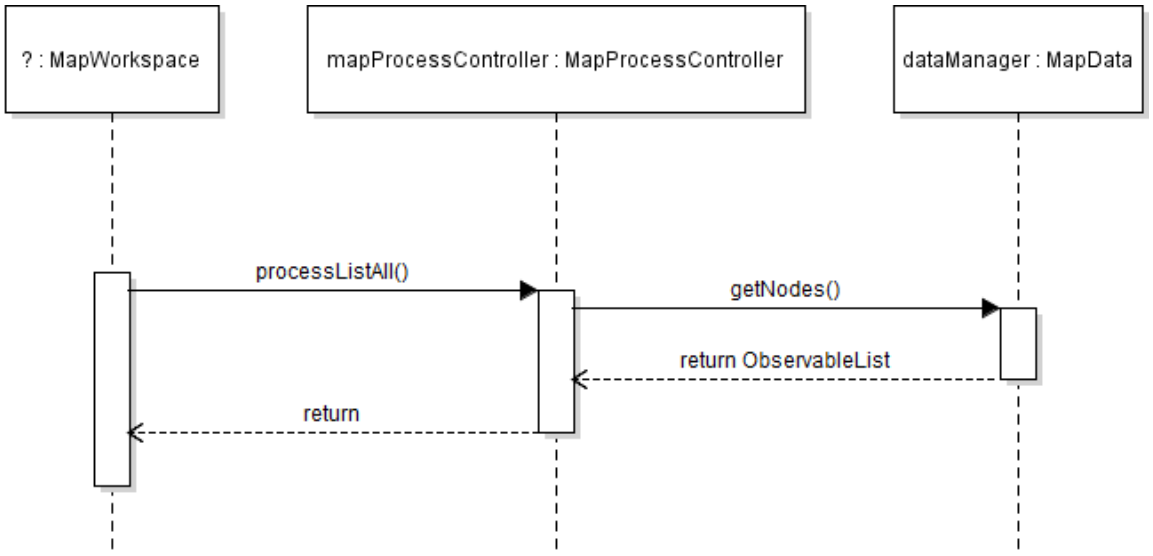
Case 2.15: Move Line End Handling



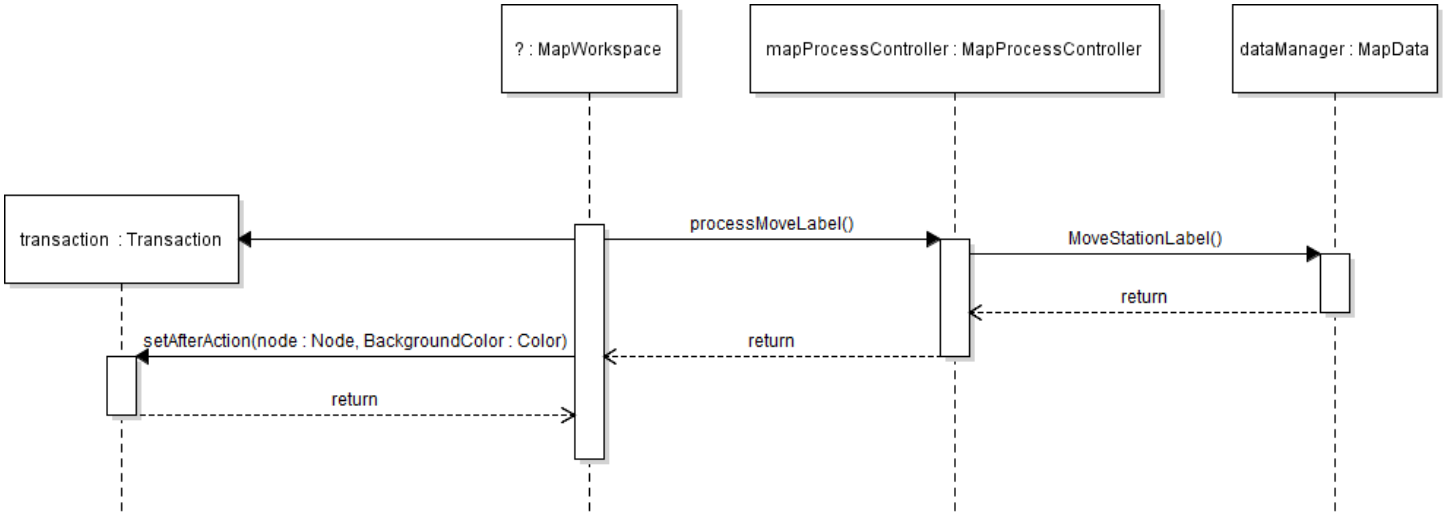
Case 2.16: Add Stations to Line Handling



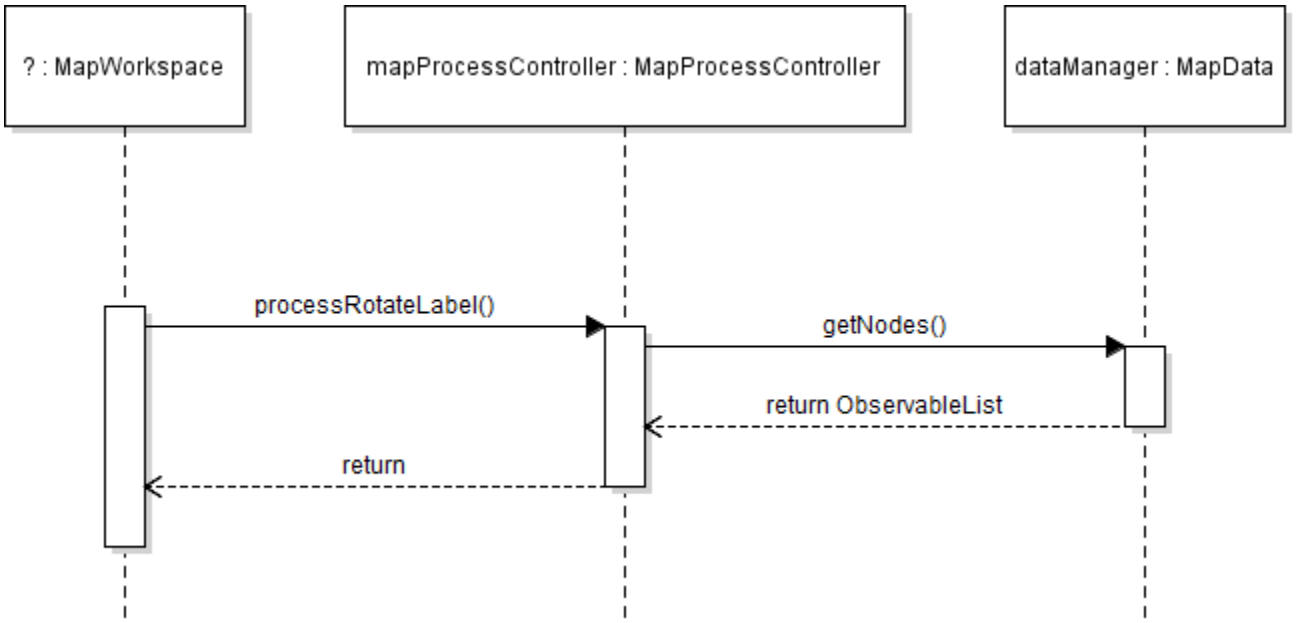
Case 2.17: Remove Stations from Line Handling



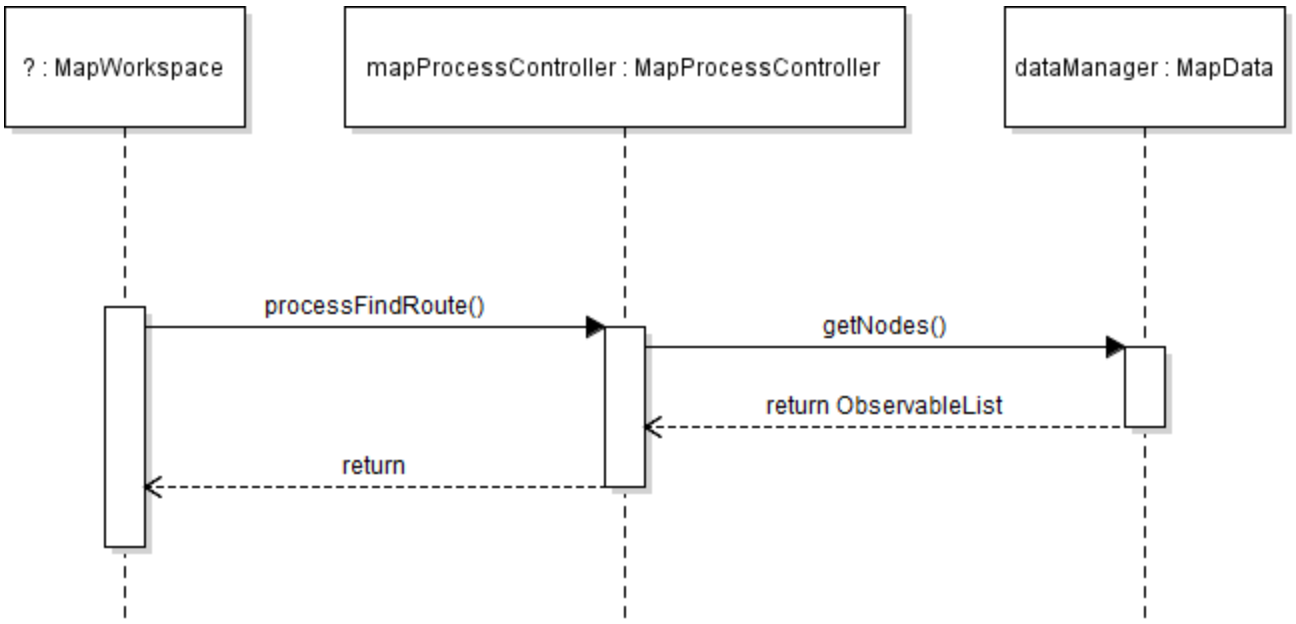
Case 2.18: List All Stations in Line Handling



Case 2.23: Move Station Label Handling



Case 2.24: Rotate Station Label Handling

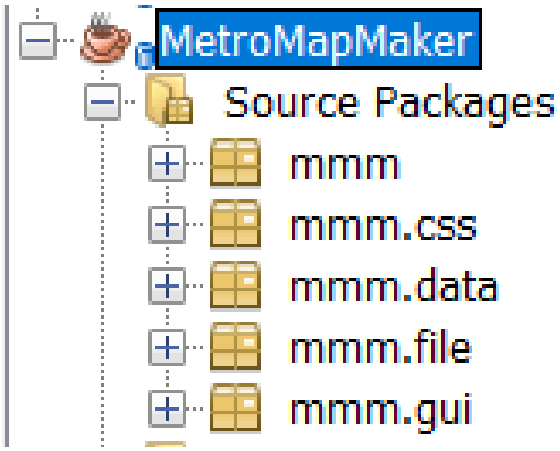


Case 2.27: Find Route Handling

5. File Structure and Formats

Note that the Desktop Java Framework and jTPS will be provided outside MetroMapMaker package, DesktopJavaFramework will be inside another package and same for jTPS. This should be imported into the necessary project for the MetroMapMaker application. Note that all necessary data and art files must accompany this program. Formatting is done in JSON as a file.

5.1 File Structure



5.2 JSON Format

```
static final String JSON_BG_COLOR = "background_color";
static final String JSON_RED = "red";
static final String JSON_GREEN = "green";
static final String JSON_BLUE = "blue";
static final String JSON_ALPHA = "alpha";
static final String JSON_NODES = "NODES";
static final String JSON_NODE = "NODE";
static final String JSON_TYPE = "type";
static final String JSON_LINE = "line";
static final String JSON_LINES = "on_lines";
static final String JSON_NAME = "name";
static final String JSON_IMAGE = "image";
static final String JSON_TEXT = "text";
static final String JSON_TEXT_FONT = "font";
static final String JSON_TEXT_SIZE = "size";
static final String JSON_TEXT_BOLD = "bold";
static final String JSON_TEXT_ITALIC = "italic";
static final String JSON_X = "x";
static final String JSON_Y = "y";
static final String JSON_WIDTH = "width";
static final String JSON_HEIGHT = "height";
static final String JSON_FILL_COLOR = "fill_color";
static final String JSON_LINE_THICKNESS = "line_thickness";
static final String JSON_LINE_COLOR = "line_color";
static final String JSON_STATION_RADIUS = "station_radius";
static final String JSON_STATION_COLOR = "station_color";

static final String DEFAULT_DOCTYPE_DECLARATION = "<!doctype html>\n";
static final String DEFAULT_ATTRIBUTE_VALUE = "";
```

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

6.2 Appendixes

N/A