MapEditor

Software Design Description

Author: Yizhi Yang
Jun, 2016
Version 1.0

Abstract: This document describes the software design for MapEditor, which is a map editor for a specified game.

Based on IEEE Std 1016™-2009 document format

1 Introduction

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

1.1 Purpose

This document is to serve as the blueprint for the construction of the MapEditor 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 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.

SimpleAppFramework – The software framework to be developed in tandem with the MapEditor such that similarly additional application can easily be constructed.

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 designing software graphically.

1.3 References

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

MapEditor SRS – YIZHI YANG's Software Requirements Specification for the MapEditor application.

1.4 Overview

This Software Design Description document provides a working design for the MapEditor software application as described in the MapEditor 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 both the **MapEditor** application and the **SimpleAppFramework** to be used in its 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 MapEditorApp

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

Figure 2.1: Design Packages Overview

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

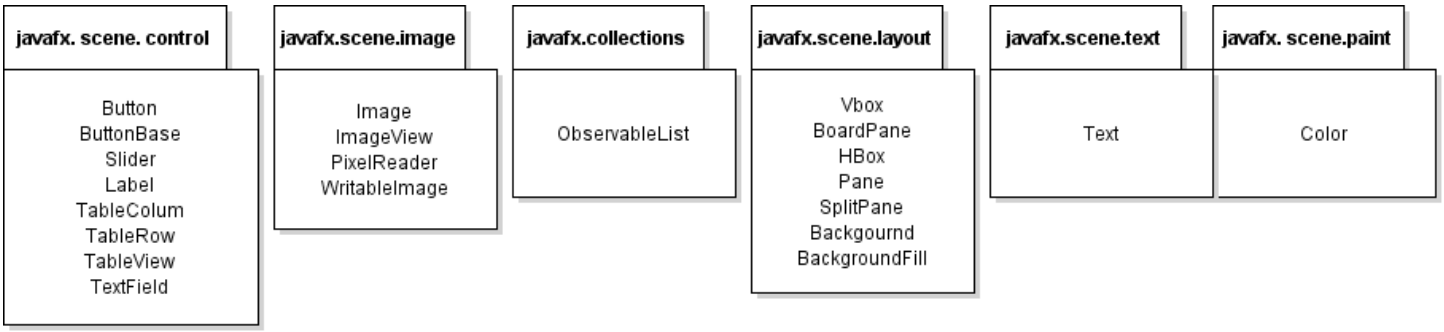


Figure 2.2: Java API Classes and Packages To Be Used

3 Class-Level Design Viewpoint

As mentioned, this design will encompass both the MapEditor application and SimpleAppFramework. 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.

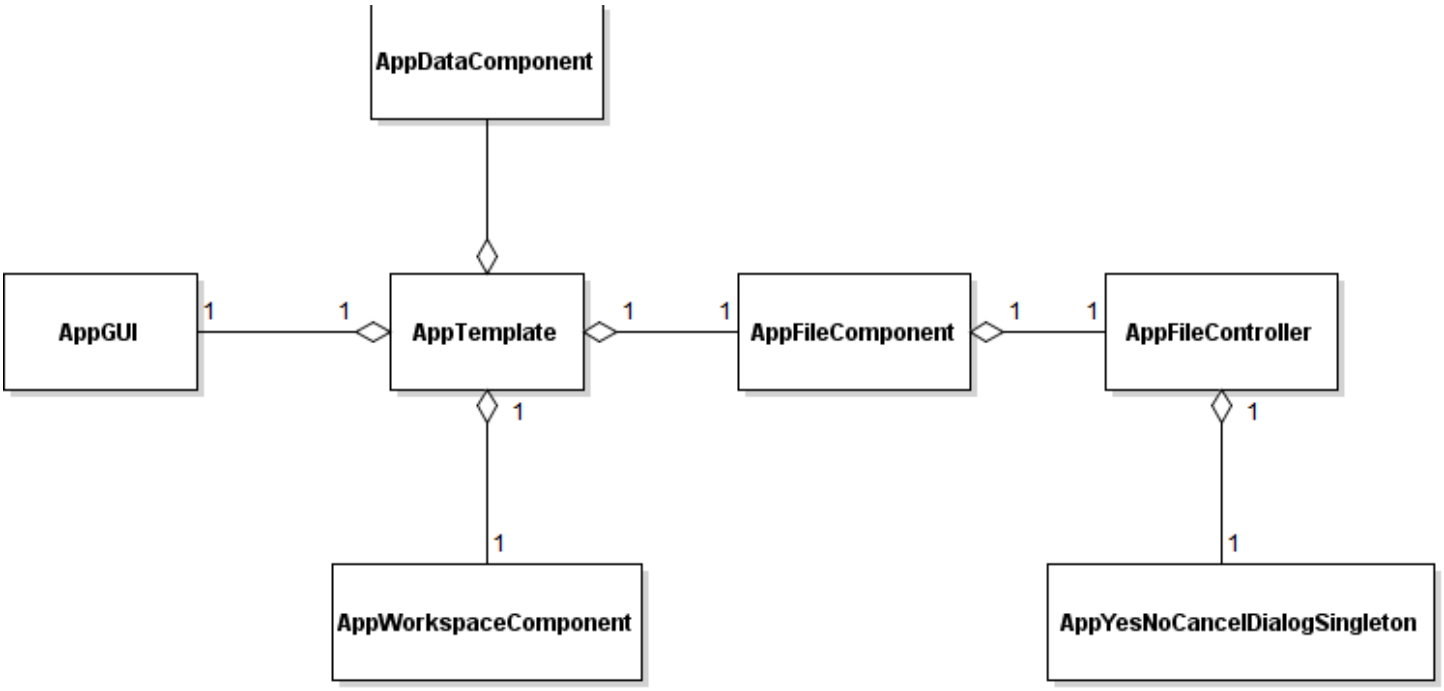


Figure 3.1: SimpleAppFramework Overview UML Class Diagram

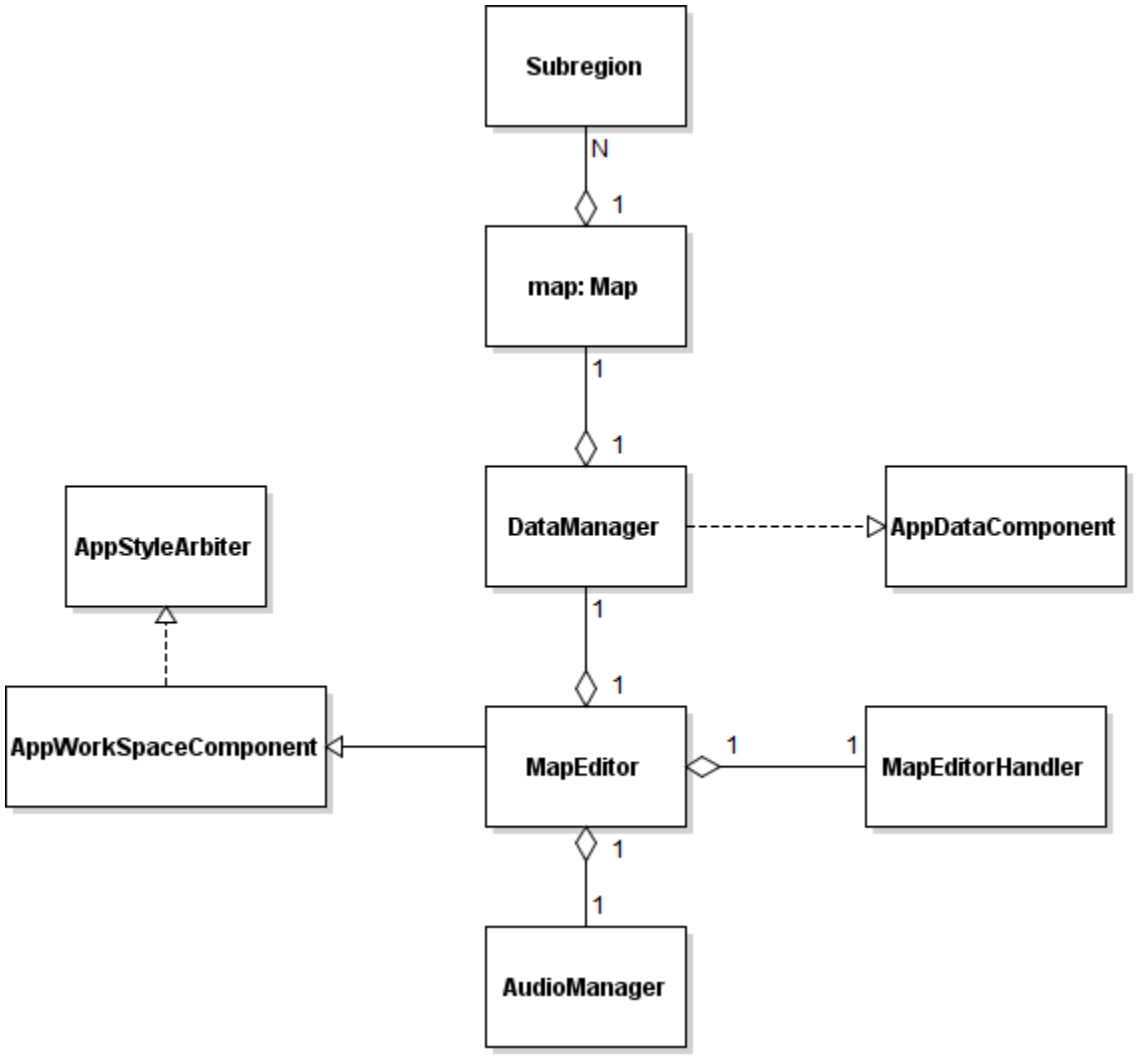


Figure 3.2: MapEditor Overview UML Class Diagram

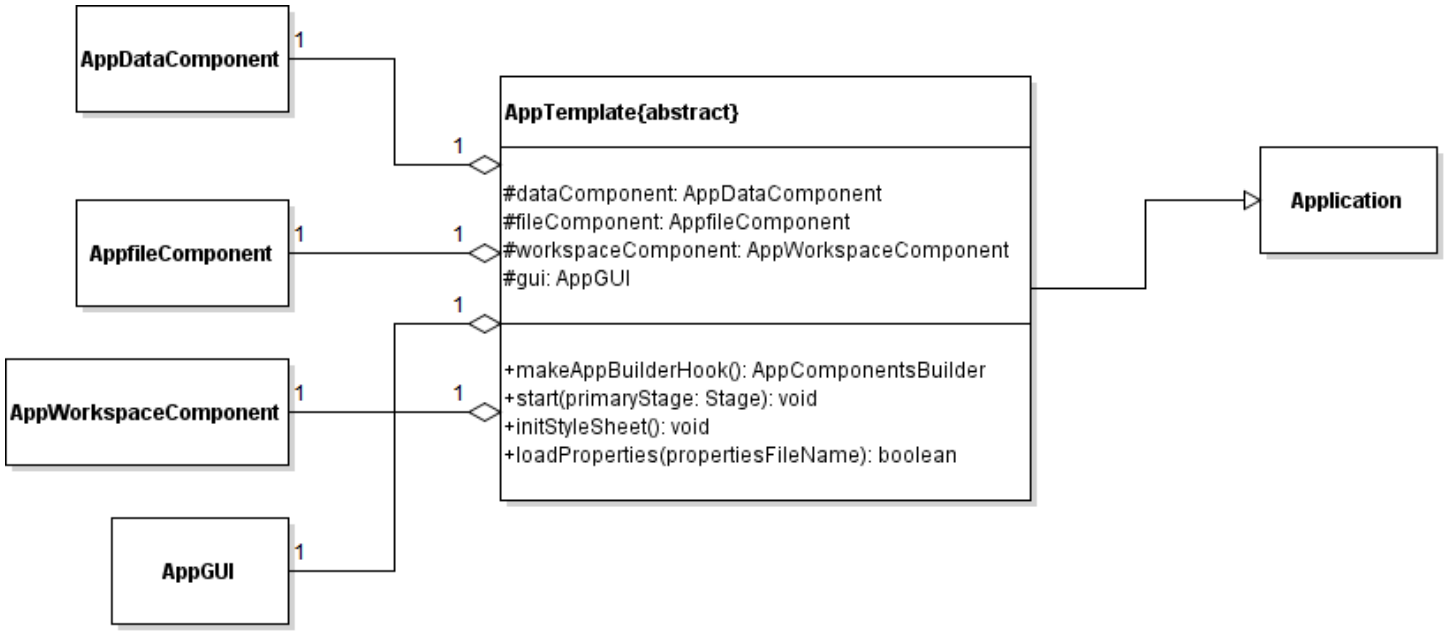


Figure 3.3: Detailed AppTemplate UML Class Diagram

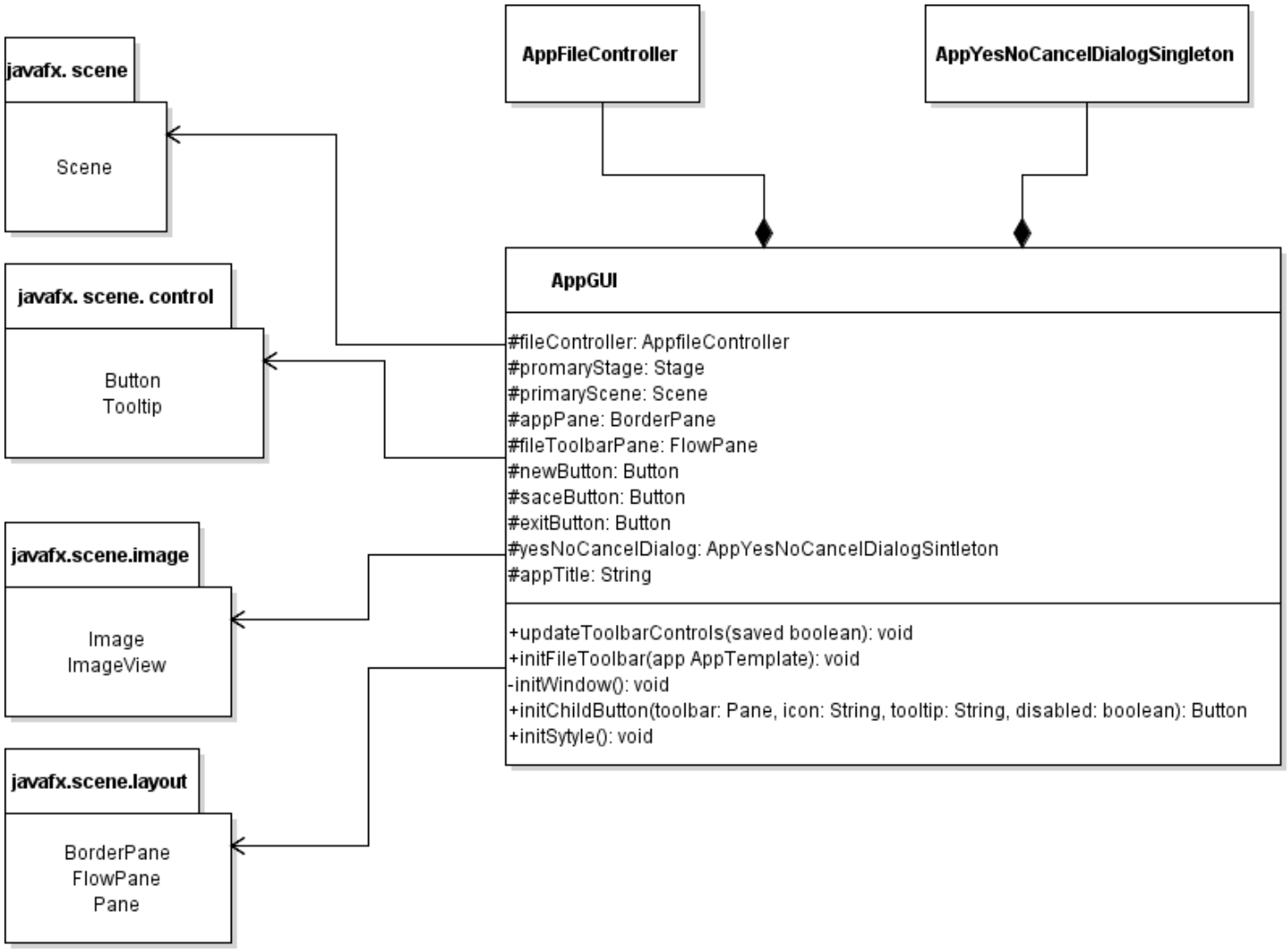


Figure 3.4: Detailed AppGUI UML Class Diagram

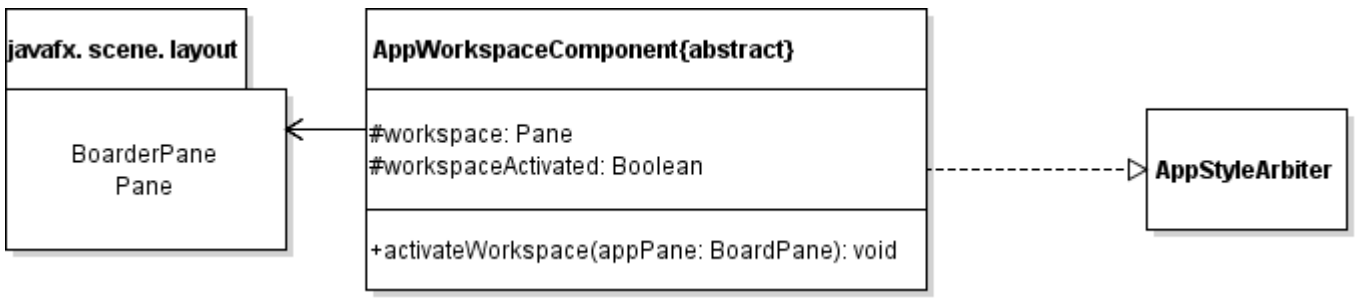


Figure 3.5: Detailed AppWorkspaceComponent UML Class Diagram

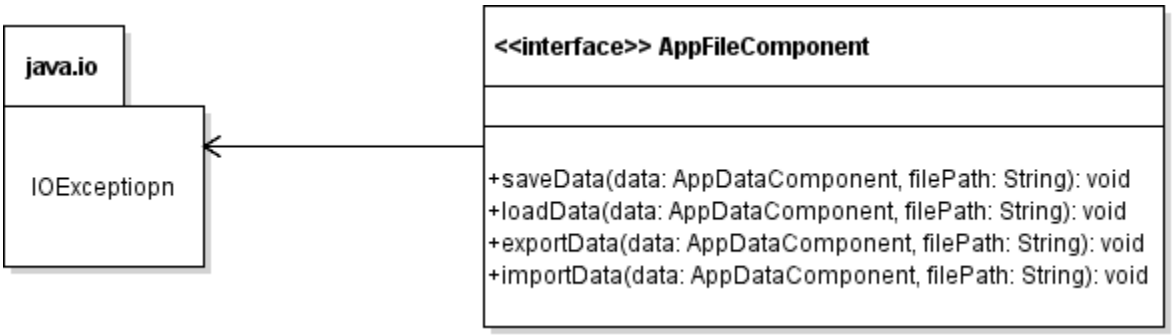


Figure 3.6: Detailed AppFileComponent UML Class Diagram

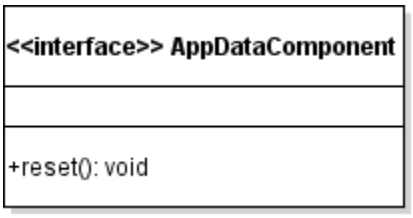


Figure 3.7: Detailed AppDataComponent UML Class Diagram

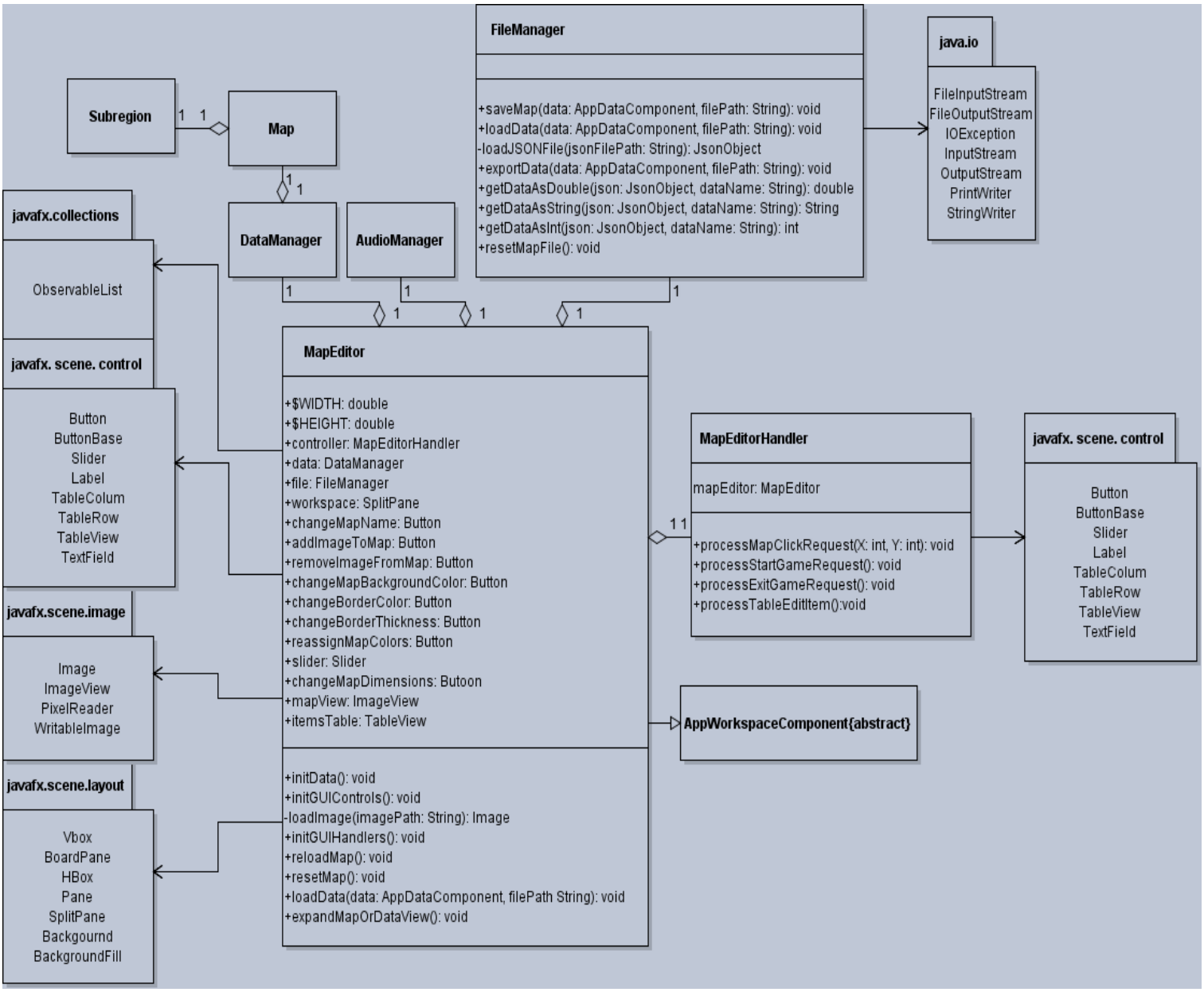


Figure 3.8: Detailed MapEditor, MapEditorHandler, and FileManager UML Class Diagram

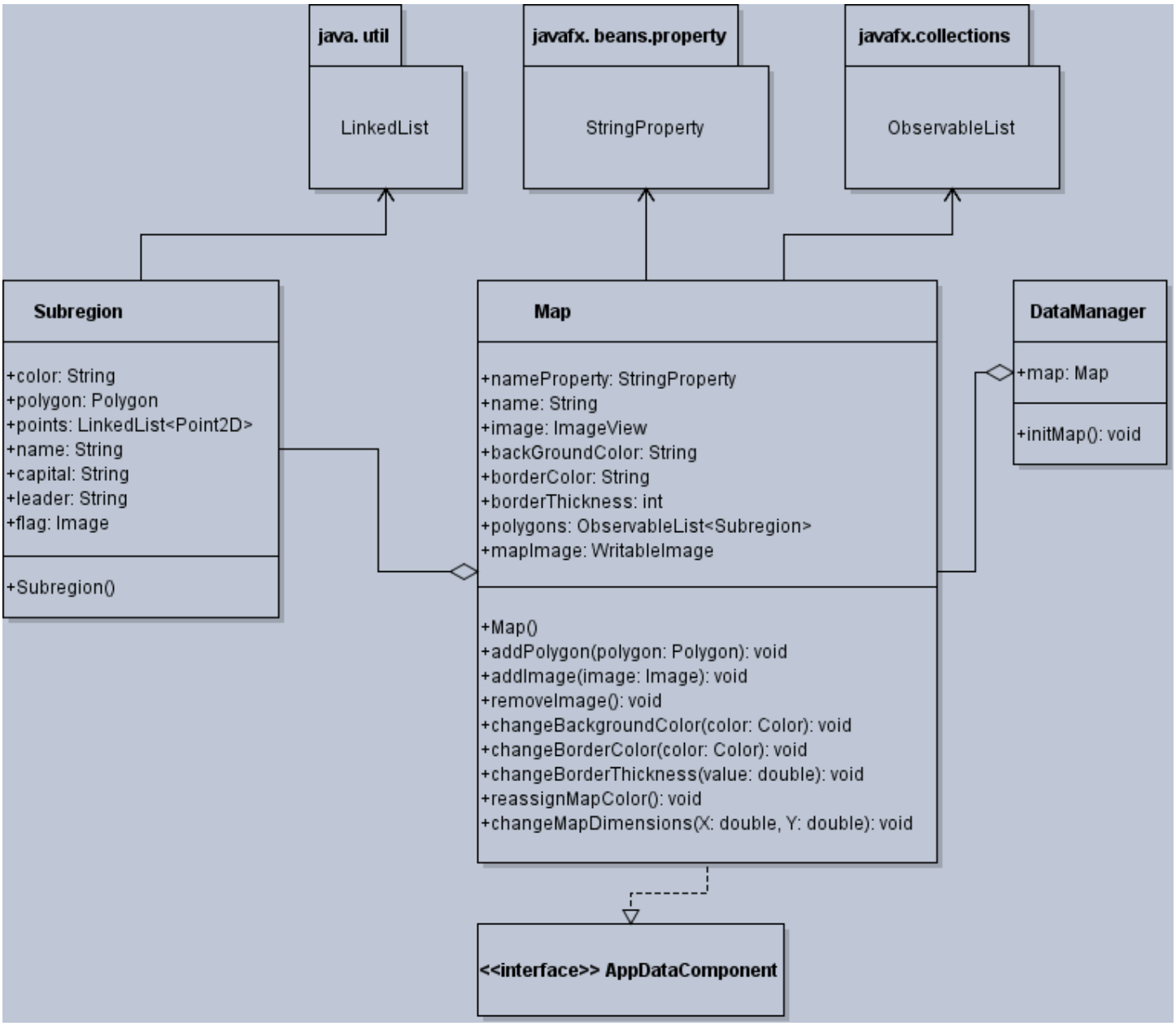


Figure 3.9: Detailed Map, and Subregion UML Class Diagram

4 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.

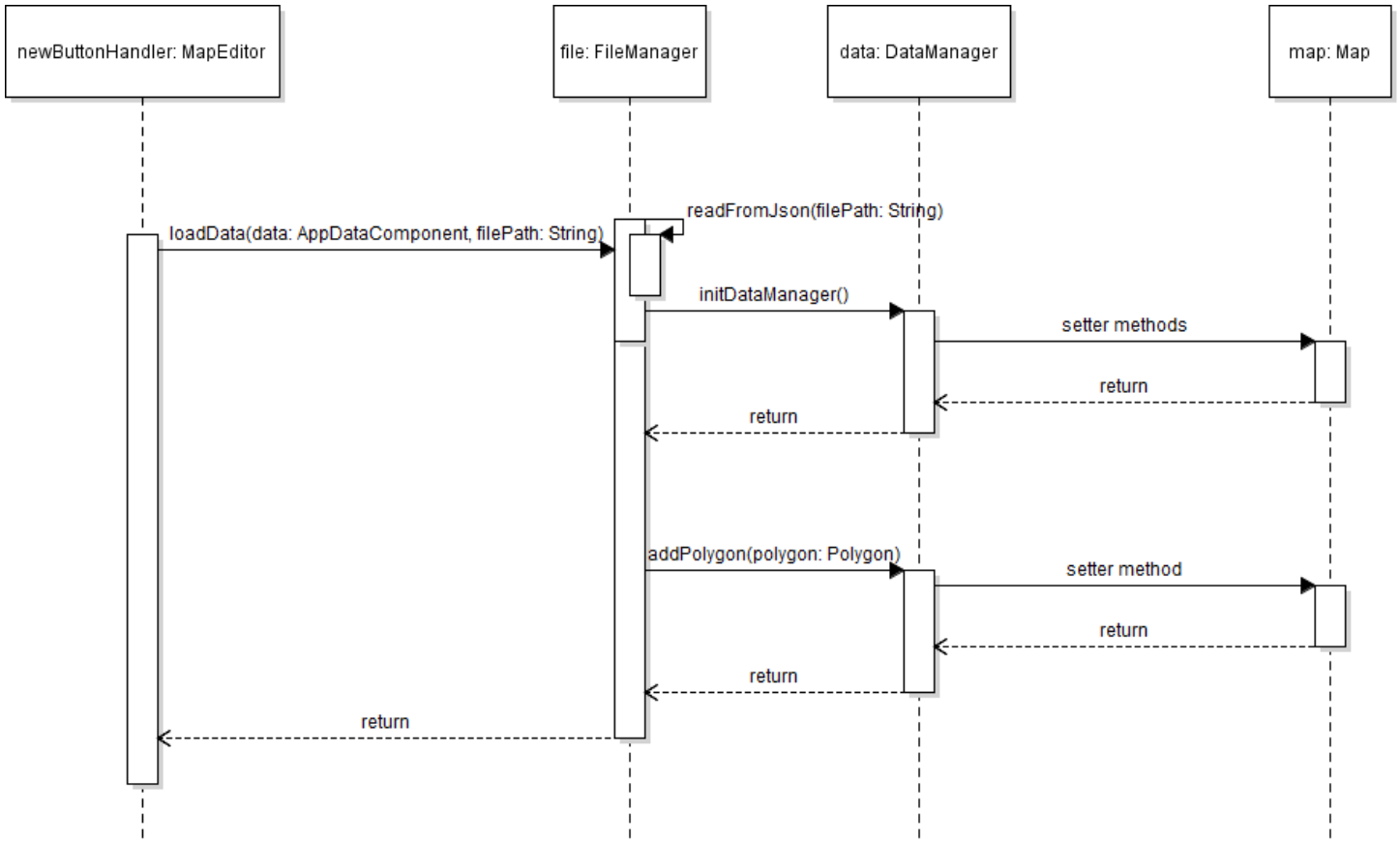


Figure 4.1: newButtonHandler UML Sequence Diagrams

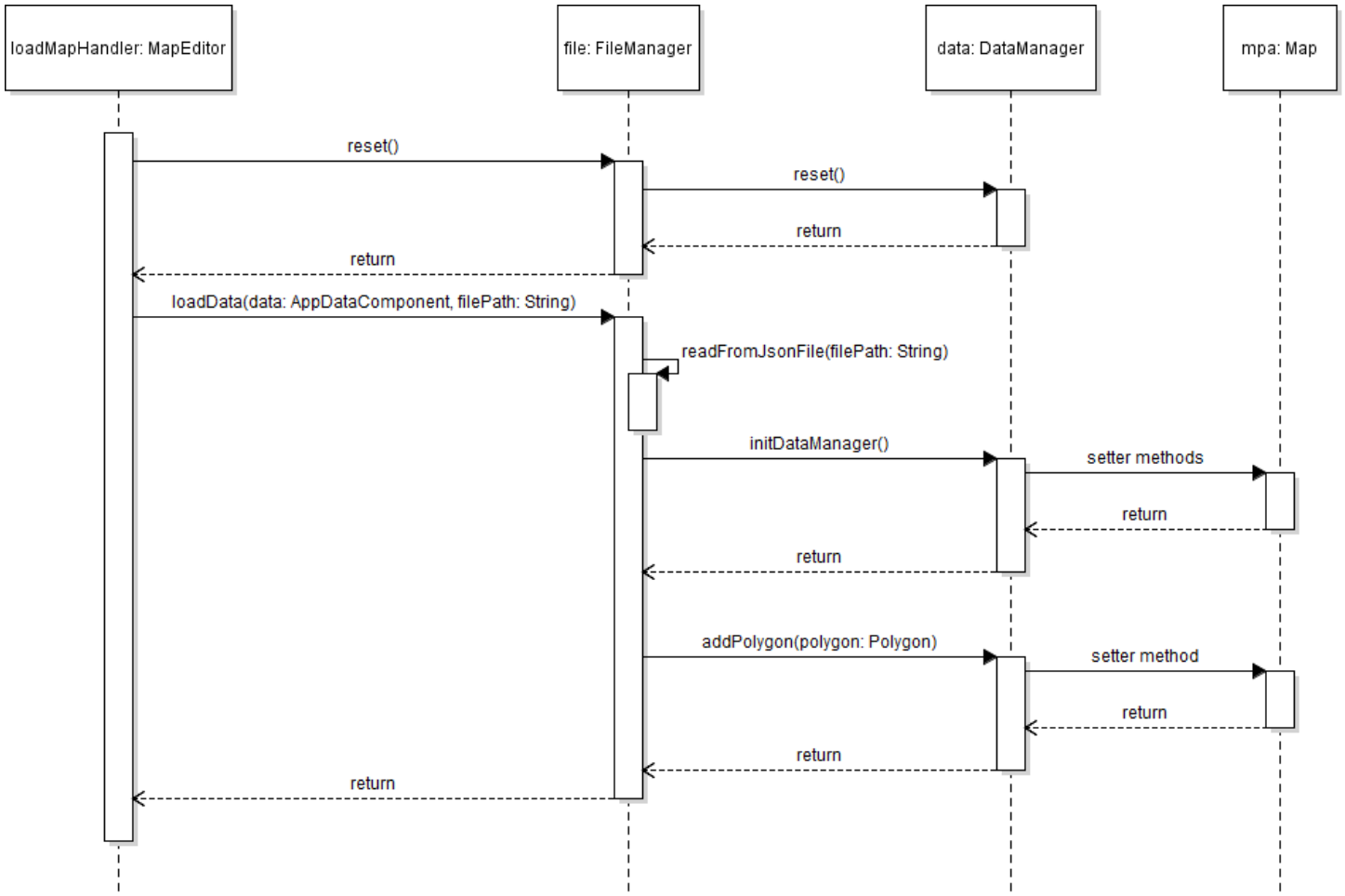


Figure 4.2: loadButtonHandler UML Sequence Diagrams

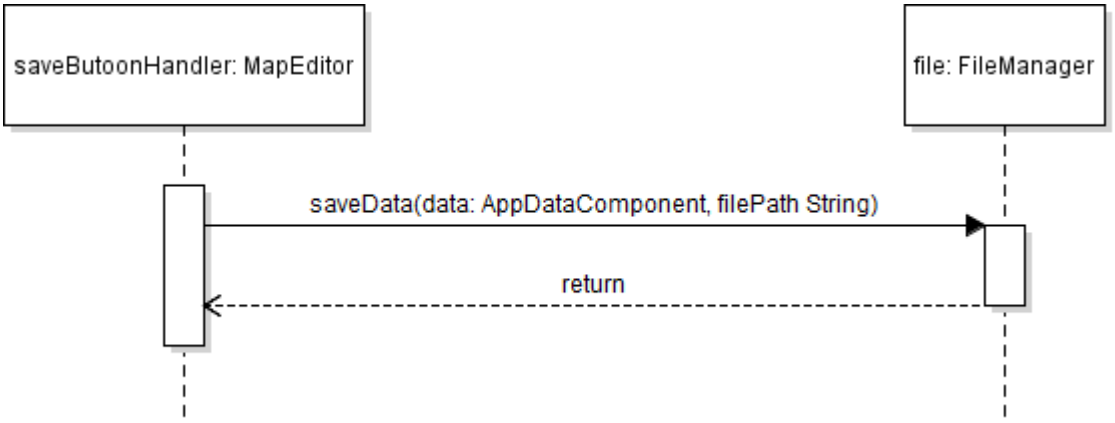


Figure 4.3: saveButtonHandler UML Sequence Diagrams

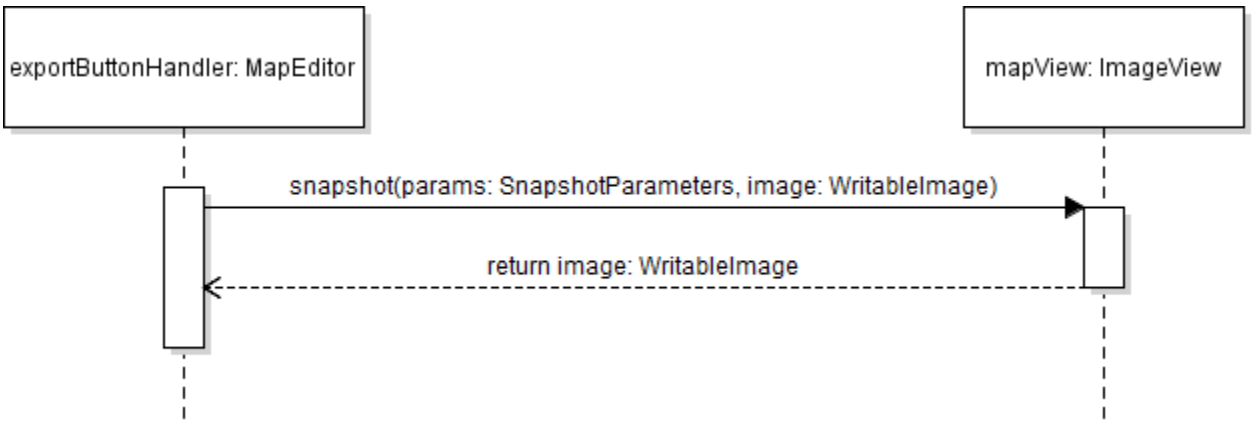


Figure 4.4: exportButtonHandler UML Sequence Diagrams

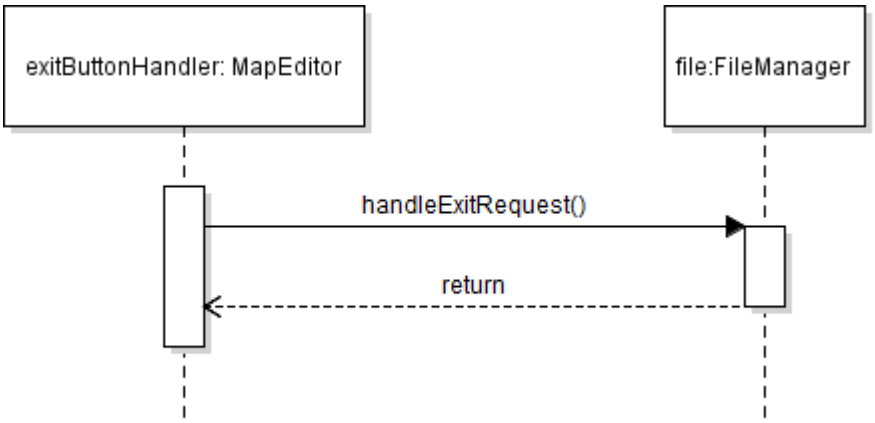


Figure 4.5: exitButtonHandler UML Sequence Diagrams

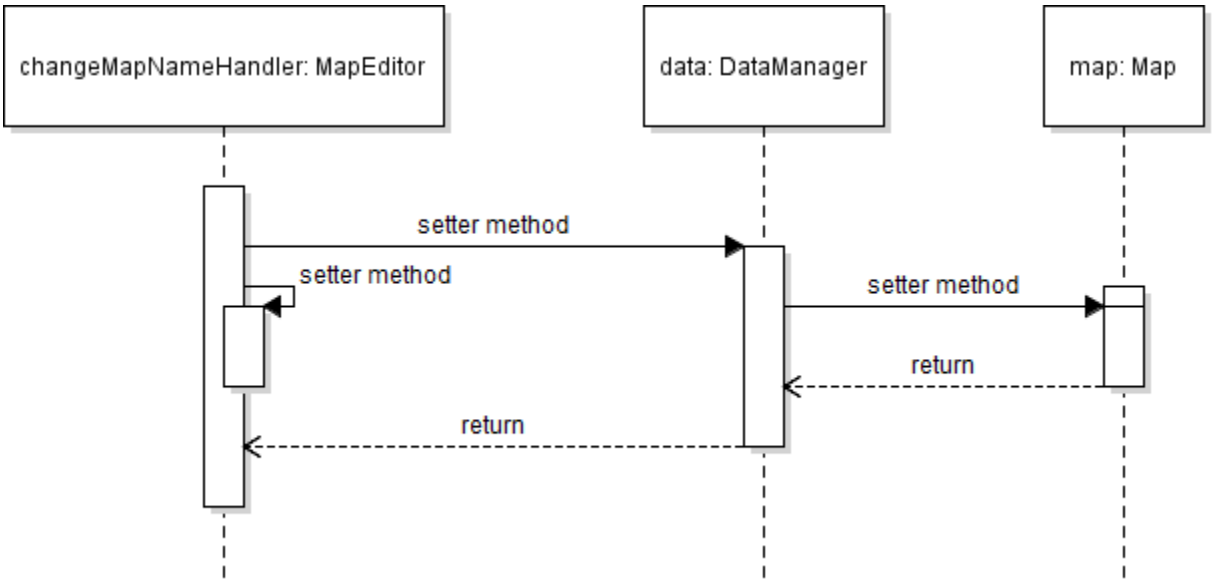


Figure 4.6: changeMapNameHandlerHandler UML Sequence Diagrams

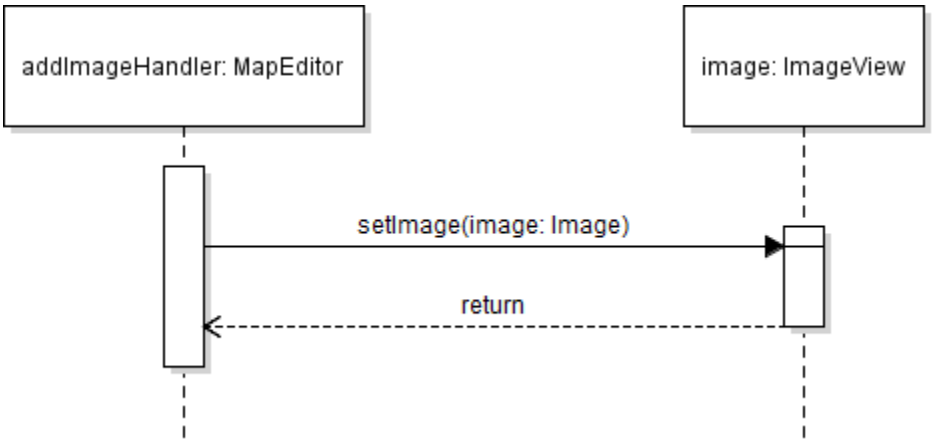


Figure 4.7: addImageHandler UML Sequence Diagrams

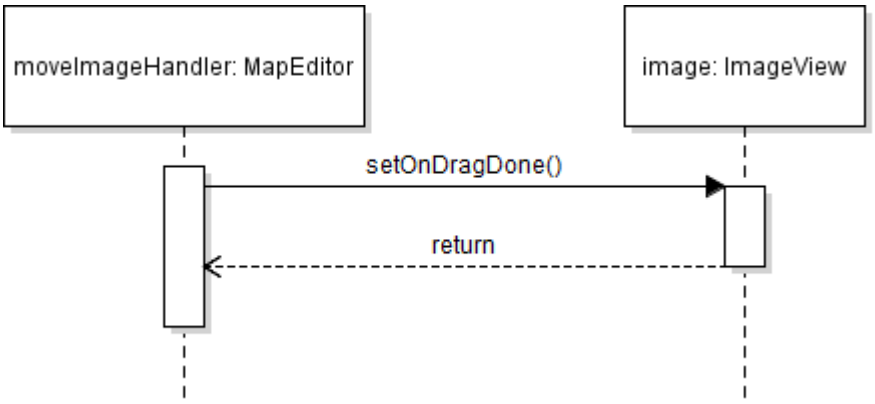


Figure 4.8: moveImageHandler UML Sequence Diagrams

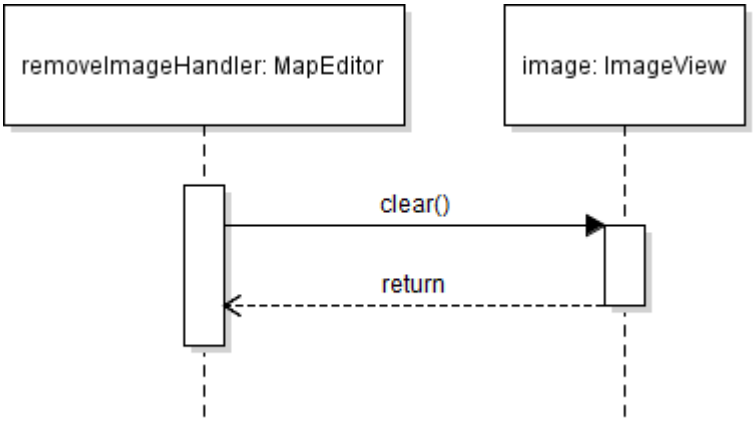


Figure 4.9: removeImageHandler UML Sequence Diagrams

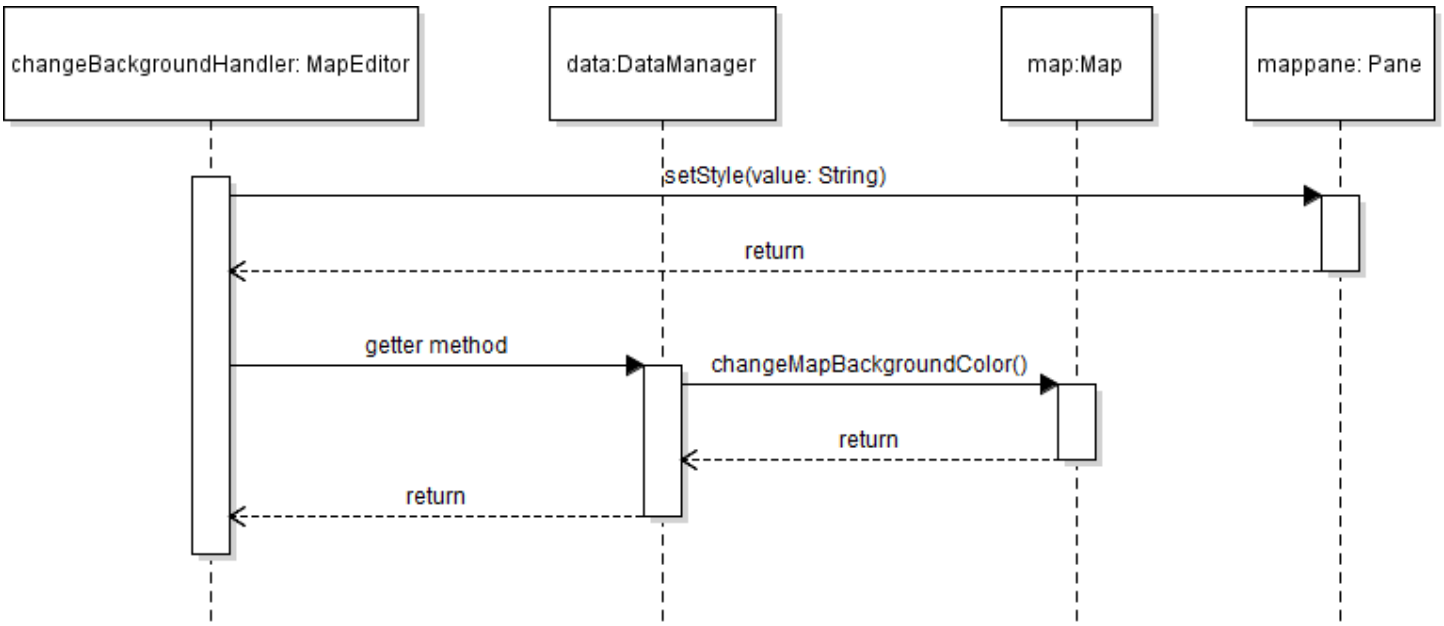


Figure 4.10: changeMapBackgroundColorHandler UML Sequence Diagrams

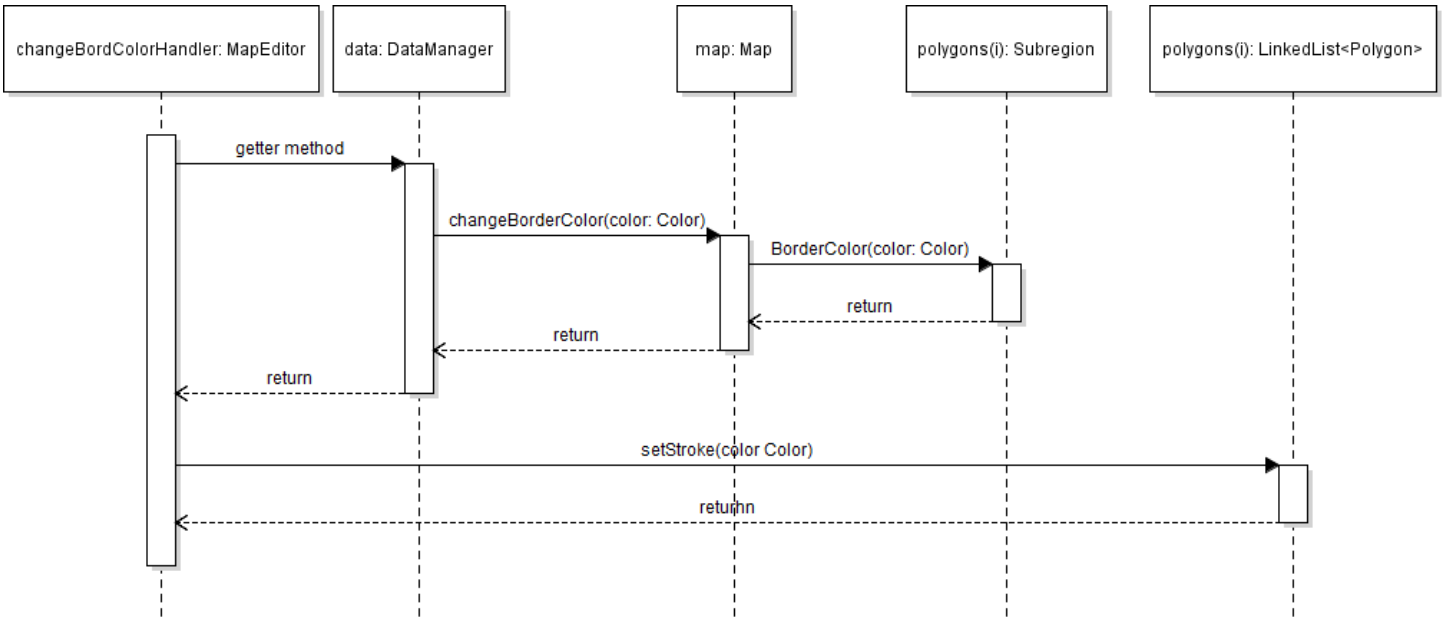


Figure 4.11: changeMapBorderColorHandler UML Sequence Diagrams

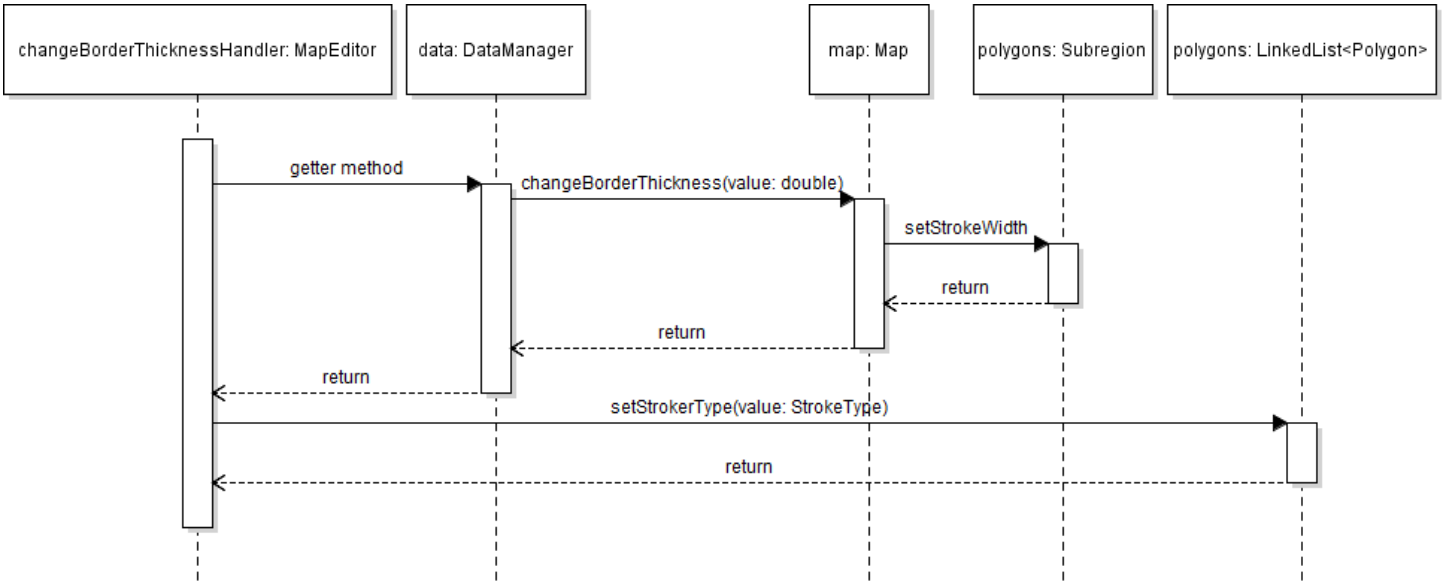


Figure 4.12: changeMapBorderThicknessHandler UML Sequence Diagrams

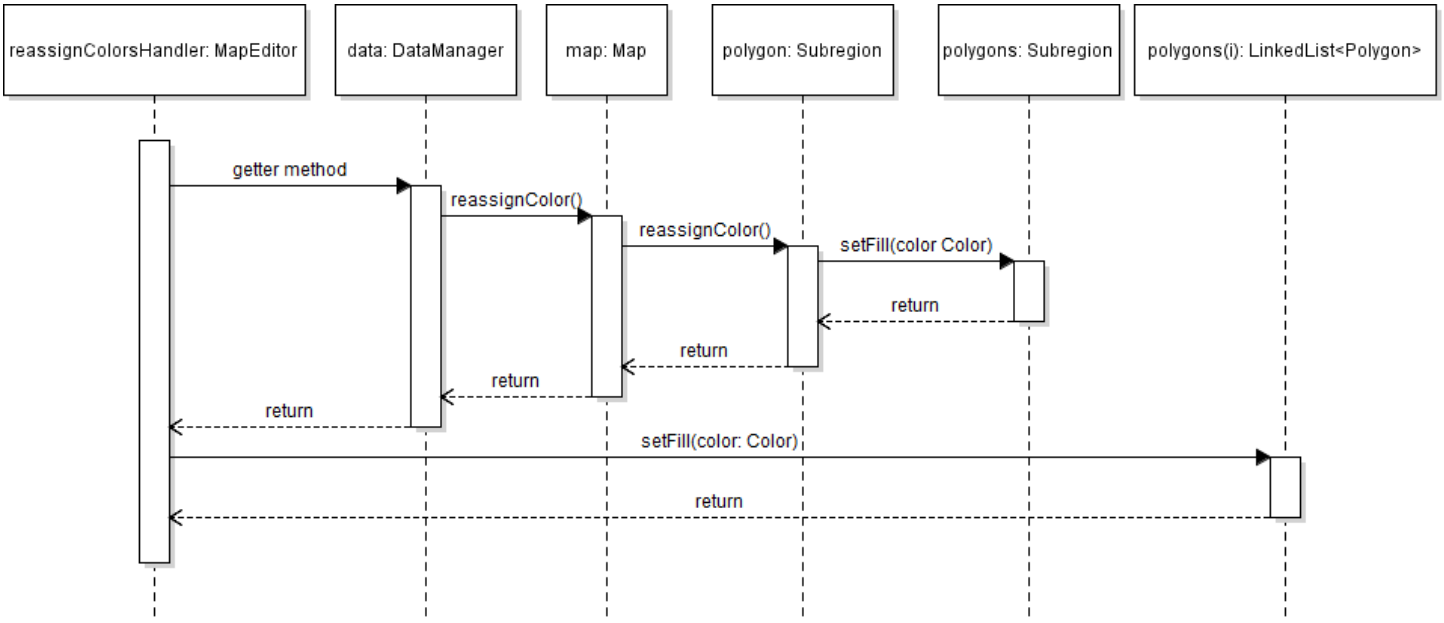


Figure 4.13: reassignMapColorsHandler UML Sequence Diagrams

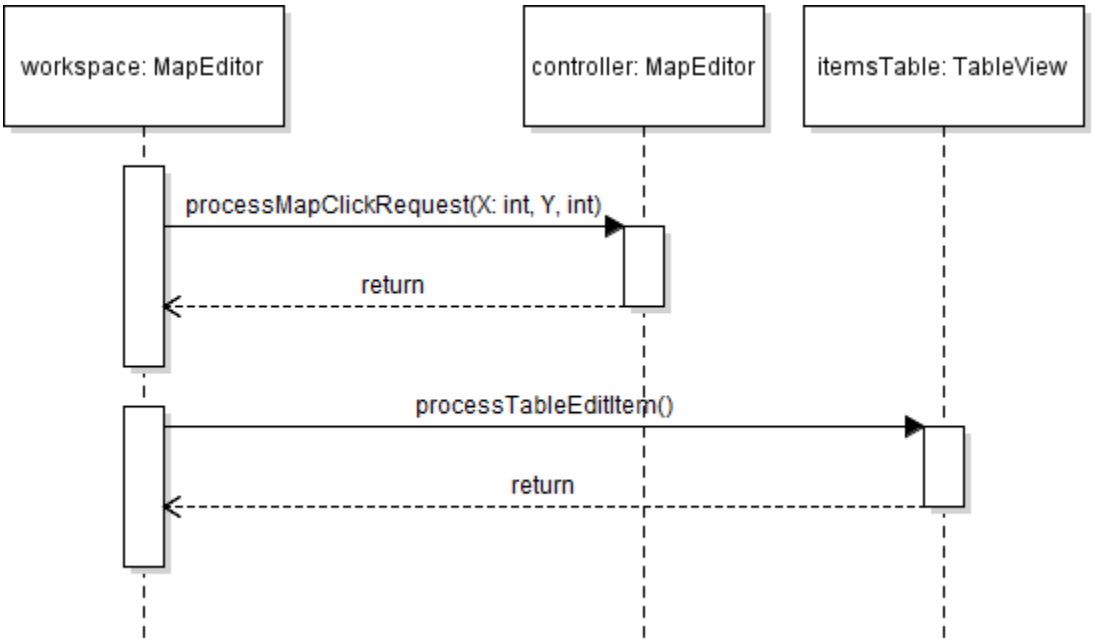


Figure 4.14: SelectSubRegionHandler UML Sequence Diagrams

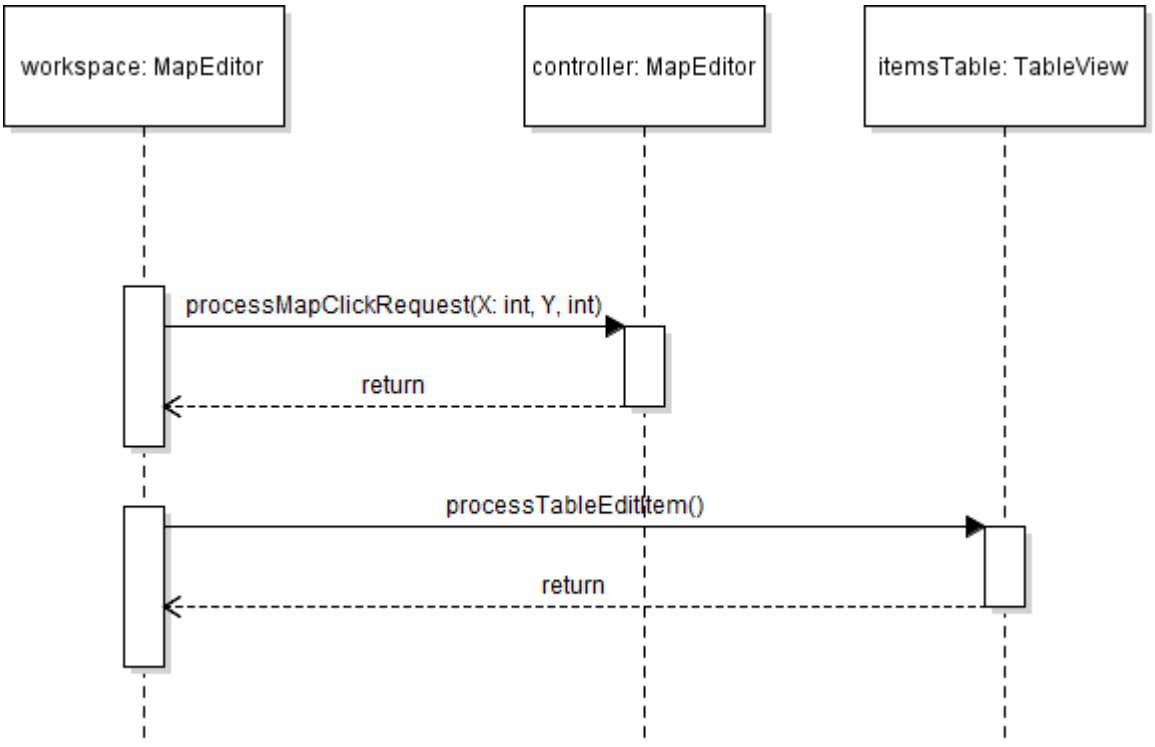


Figure 4.15: ViewSubRegionHandler UML Sequence Diagrams

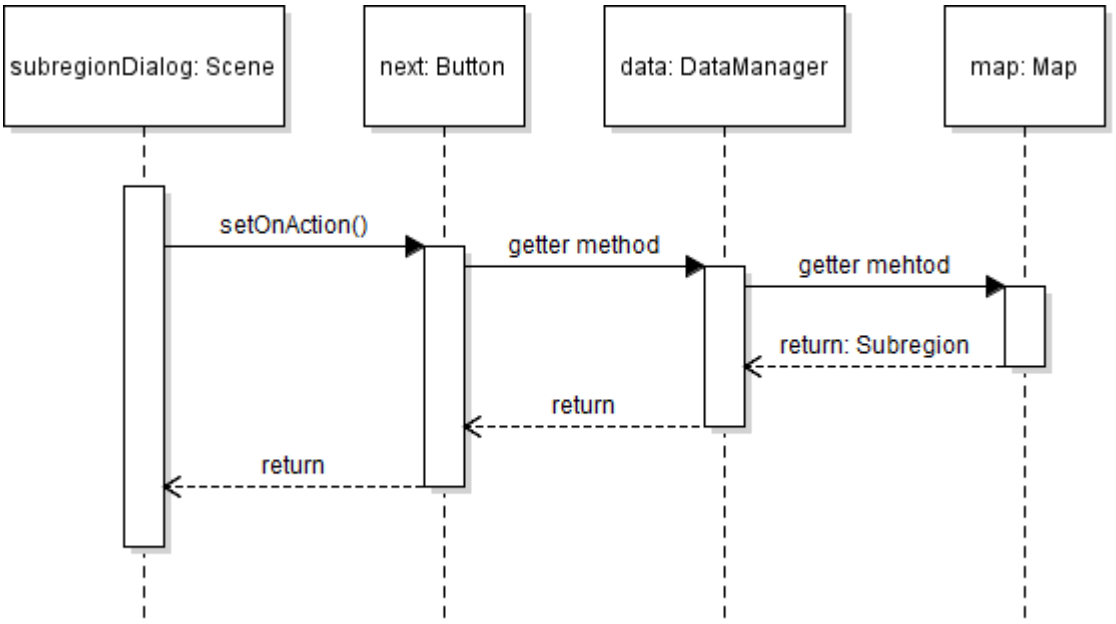


Figure 4.16: nextSubRegionHandler UML Sequence Diagrams

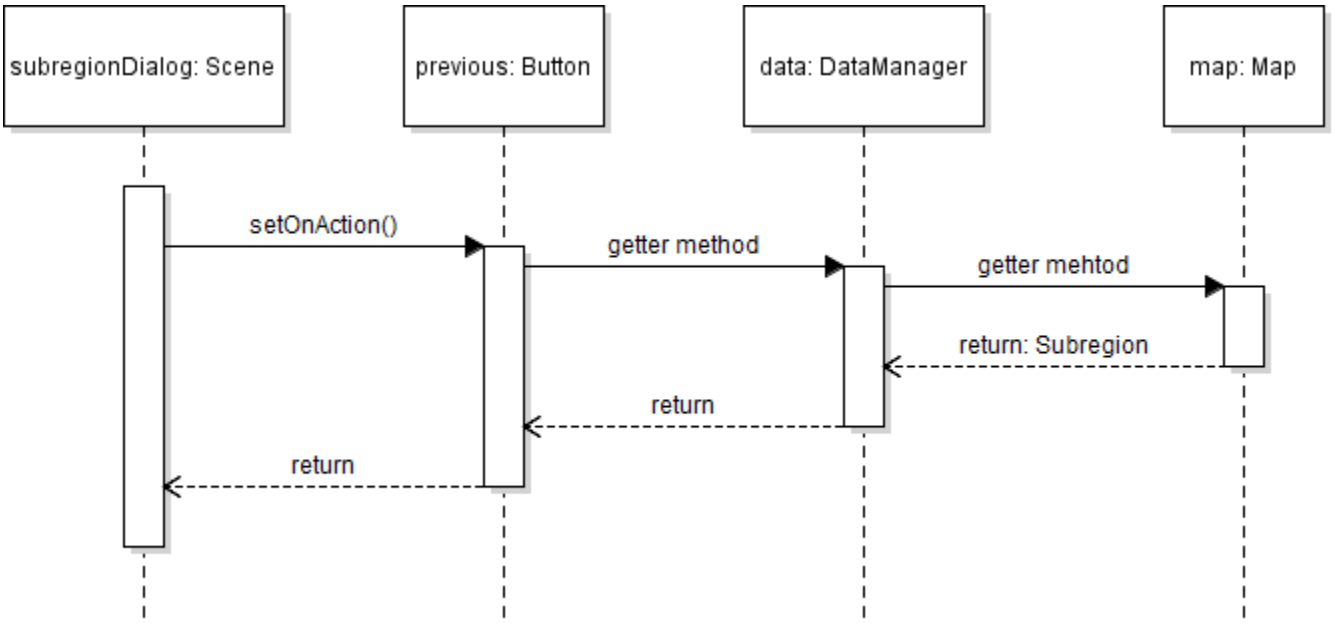


Figure 4.17: previousSubRegionHandler UML Sequence Diagrams

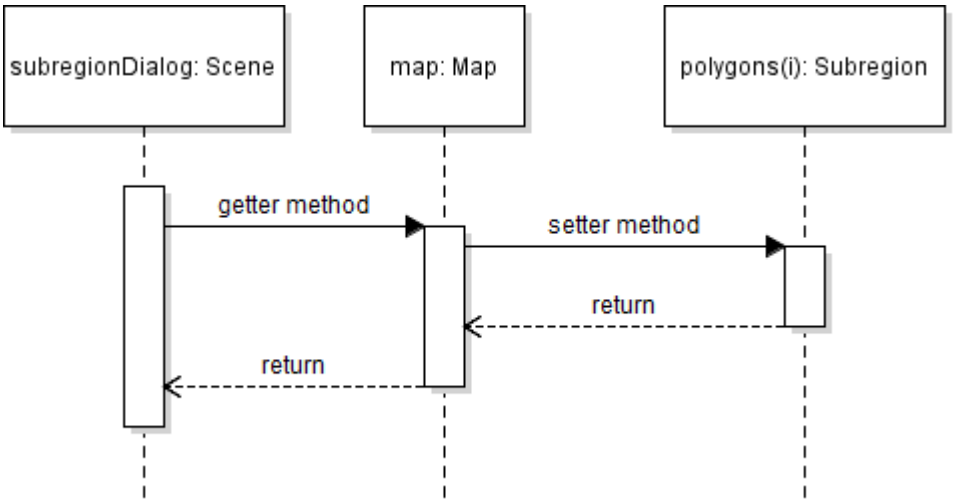


Figure 4.18: editSubRegionHandler UML Sequence Diagrams

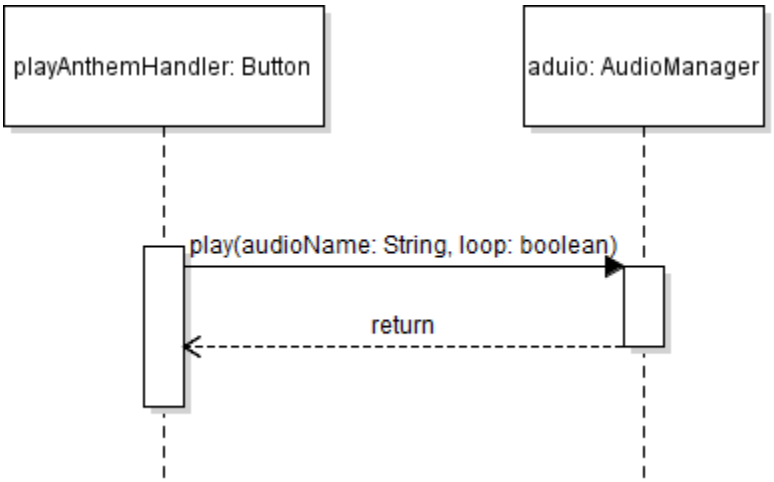


Figure 4.19: playAnthemHandlerHandler UML Sequence Diagrams

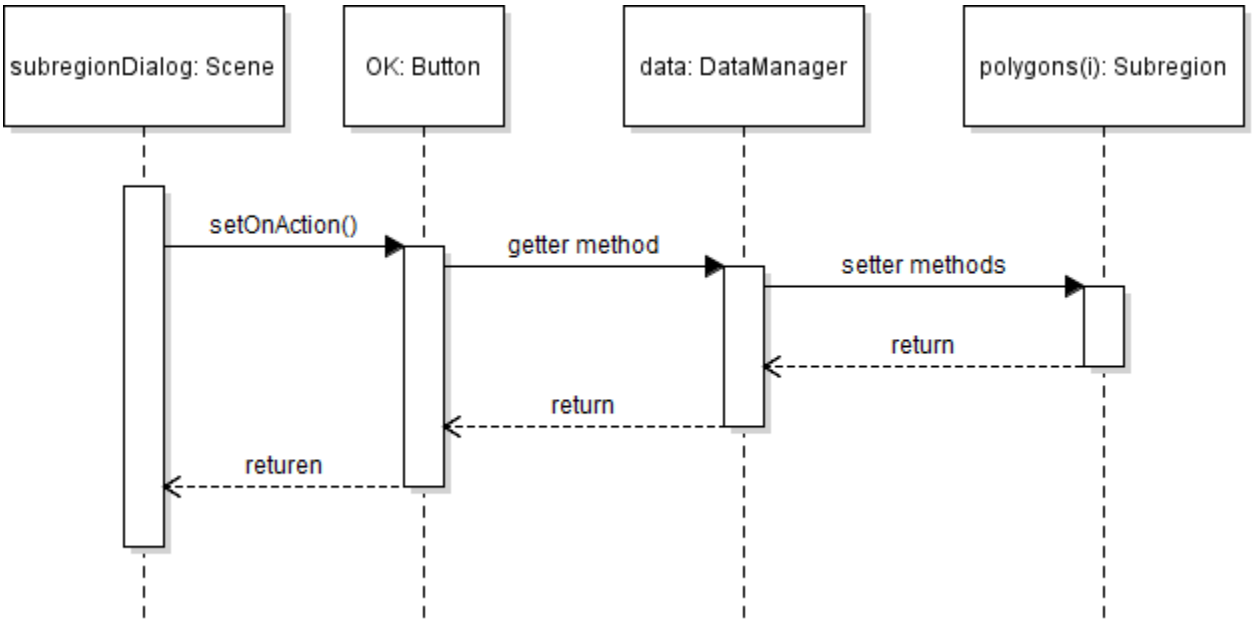


Figure 4.20: closeSubRegionDialogHandler UML Sequence Diagrams

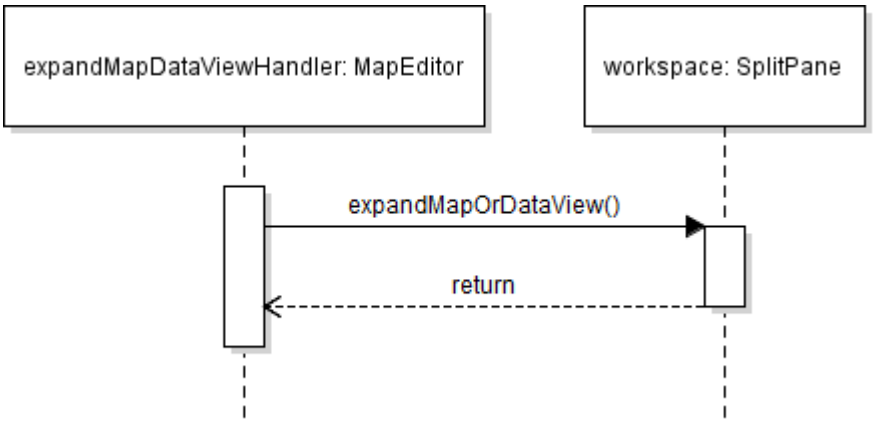


Figure 4.21: expandMap/DataViewHandler UML Sequence Diagrams

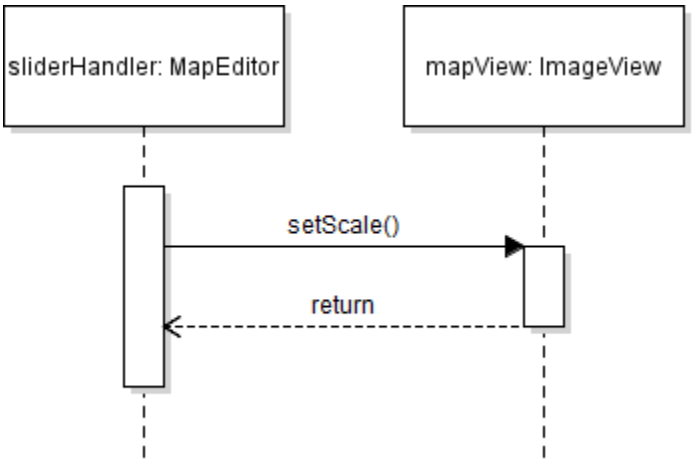


Figure 4.22: zoomMapIn/OutHandler UML Sequence Diagrams

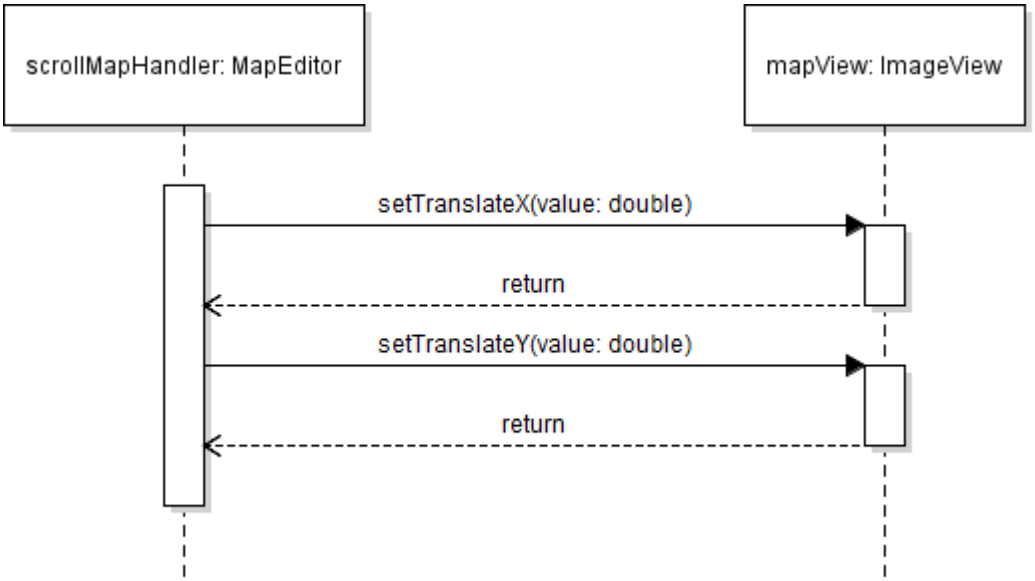


Figure 4.23: ScrollMapHandler UML Sequence Diagrams

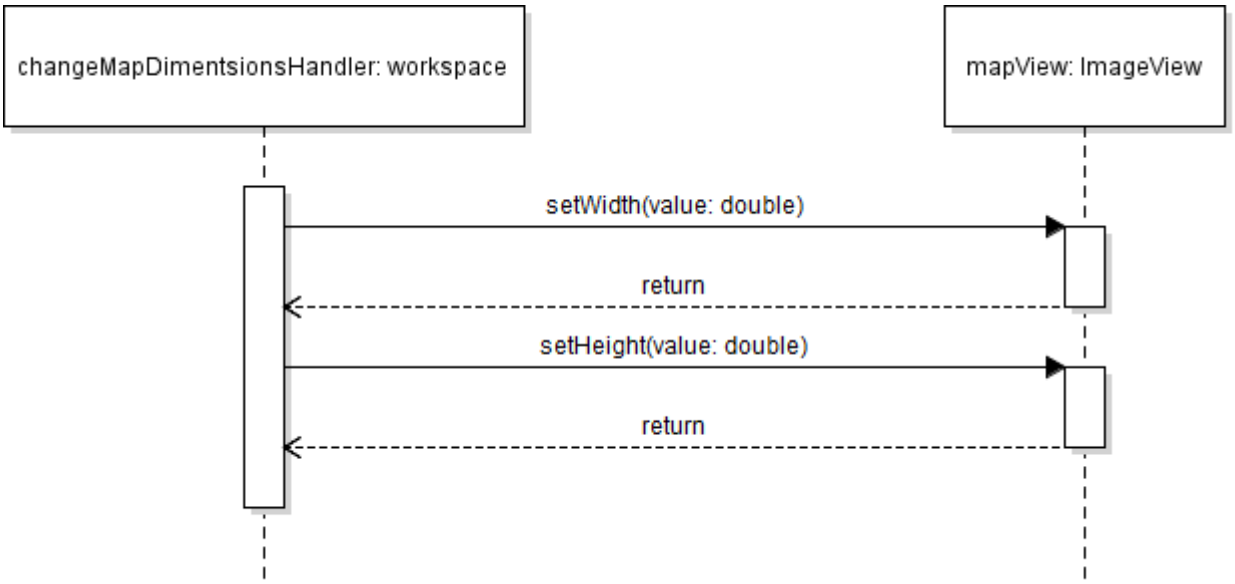
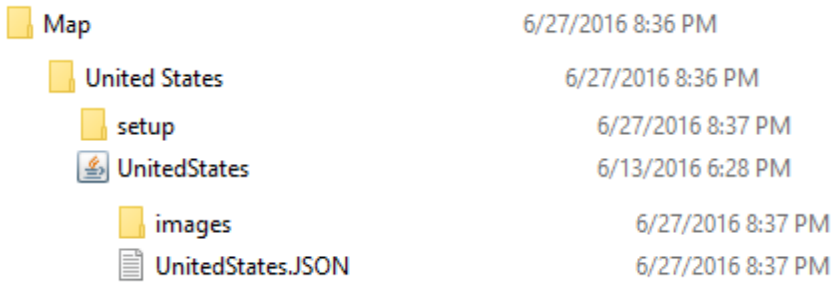


Figure 4.24: changeMapDimensionsHandler UML Sequence Diagrams

5. File Structure and Formats

Note that the SimpleAppFramework will be provided inside SimpleAppFramework.jar, a Java ARChive file that will encapsulate the entire framework. This should be imported into the necessary project for the MapEditor application and will be included in the deployment of a single, executable JAR file titled ****.jar (depends on the actual name of the map). Note that all necessary data and 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 images directory.

5.1 Directory Structures



(In this case, Map is the parent Directory. United States contains a .jar file and all the needed information. Inside of the United States.JSON file, a specified format is used to store all the information of the United States, such as a reference of the geographical coordinates of the United States, the background color of the map, and a scale value of the map. All necessary images should go in the images directory.)

Figure 5.2: Data File Formats

The UnitedStates.JSON provides the reference of geographical coordinates of the map, the properties of the map, and the subregions' properties in this application. The format is:

{
"referenceOfSubregions": reference of the subregion file (for example: Map\Coordinates\UnitedStates),
"name": name of the map (for example: United States),
"backgroundColor": background color of the map (for example: #FFFF00"),
"zoomInLevel": zoom in level of the map (for example: 1),
"referenceOfImage":reference of the image reference of the image(for example: Map\UnitedStates\setup\images),
"dimensionX": X (for example: 500.0),
"dimensionY": Y (for example: 600.0),
"polygons": referencefOfSubregions,
}

The follow format is used to store each individual subregion

"referenceOfGeographicalCoordiates": reference of the coordinate file,
"subregions": [
{
"color": filledColor,
"name": name of the subregion,
"capital": capital of the subregion,
"leader": name of the leader,
"leaderImage": referenceOfTheLeaderImage,
"flage": referenceOfTheFlage,
"anthem": referenceOfTheAnthem
},
{
"color": filledColor,
"name": name of the subregion,
"capital": capital of the subregion,
"leader": name of the leader,
"leaderImage": referenceOfTheLeaderImage,
"flage": referenceOfTheFlage,
"anthem": referenceOfTheAnthem
},
.
.
.
.

]

5.1 Directory Structures

Additional files are needed in this application. Such as a file that stores the geographical coordinates, and a folder that stores flag images and national anthems. These files can be referenced by JSON files (the formats are specified in 5.2)