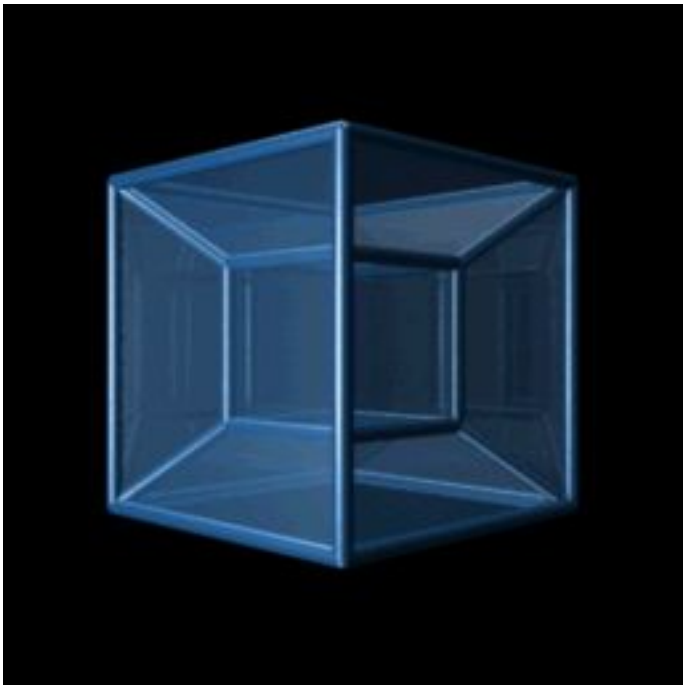
# *Data Visualization Application(Data-ViLiJ™)*

Software Design Description



**Author:** Jeremy Herrmann
Professaur Inc.™
Created March, 2018

# 1. Table of Contents

# 2. Introduction

Within this document will be all the information regarding the design structure of the Data Visualization Application. This is the Software Design Description(SDD) for the application.

## 2.1. Purpose

The Software Design Description will provide all the necessary information for creating the Data Visualization Application. With the use of UML diagrams, we will document all the packages, classes, methods, and variables that will be used in the application's development, alongside sequence diagrams which describe the chronological process that everything occurs in. This document will be essential in describing the construction of the application as a whole and is intended for anyone who is interested in learning about the internal structure of the application and how it executes.

## 2.2. Scope

This Data Visualization Application will serve as an application for algorithm visualization. Using graphs and data plotting techniques, any algorithm can be plotted and visualized. This application will be using the DataViLiJ framework in order to make the process and design simpler. As a side node, the DataViLiJ framework will not be modified throughout the design. In addition to the DataViLiJ framework, we will be utilizing the XMLUtilities library for extra functionality.

## 2.3. Definitions, acronyms, and abbreviations

1. Framework : An abstraction in which software providing generic functionality for a broad and common need can be selectively refined by additional user-written code, thus enabling the development of specific applications, or even additional frameworks. In an object-oriented environment, a framework consists of interfaces and abstract and concrete classes.
2. Graphical User Interface (GUI) : An interface that allows users to interact with the application through visual indicators and controls. A GUI has a less intense learning curve for the user, compared to text-based command line interfaces. Typical controls and indicators include buttons, menus, check boxes, dialogs, etc.

3. IEEE: Institute of Electrical and Electronics Engineers, is a professional association founded in 1963. Its objectives are the educational and technical advancement of electrical and electronic engineering, telecommunications, computer engineering and allied disciplines.
4. Instance : A 2-dimensional data point comprising a x-value and a y-value. An instance always has a name, which serves as its unique identifier, but it may be labeled or unlabeled.
5. Software Design Description (SDD) : A written description of a software product, that a software designer writes in order to give a software development team overall guidance to the architecture of the project.
6. Software Requirements Specification (SRS) : A description of a software system to be developed. It lays out functional and non-functional requirements and may include a set of use cases that describe user interactions that the software must provide. This document, for example, is a SRS.
7. Unified Modeling Language (UML) : A general-purpose, developmental modeling language to provide a standard way to visualize the design of a system.
8. Use Case Diagram : A UML format that represents the user's interaction with the system and shows the relationship between the user and the different use cases in which the user is involved.
9. User : Someone who interacts with the DataViLiJ application via its GUI.
10. User Interface (UI) : See Graphical User Interface (GUI).

## 2.4. References

**DataViLiJ™SRS** - Professaur Inc Software Requirement Specification for the Data Visualization Application.

## 2.5. Overview

This Software Design Description will be providing a working internal structural design for the data visualization application by using the data-vilij framework. The first section that is apparent is the Table of Contents for the document. Following that, section two is a general overview and introduction to the application as a whole where we define some vocabulary and acronyms, state the purpose of the application, and provide the material in which we referenced. Section three of this document will document the packages and frameworks used and designed within the application. Similarly, section four and five of this document will be providing the class level and method level viewpoints respectively. The class level viewpoint will specify which field and methods should be included in the construction of each class via UML diagrams. The

method level viewpoint will be showing the interactions between each class. Section six will cover the File and data structures alongside the formats that were used when making the application. Finally, section 7 will be the remaining supporting information.

# 3. Package-level Design Viewpoint

Within the package-level design viewpoint, the software overview for the Data-ViLiJ framework alongside the Java API will be described. Throughout the application, the Java API was frequently referenced, and in this section, the information regarding how they will be used can be found.

## 3.1. Software Overview

The Data Visualization Application will make full use of the ViLiJ package without changing anything. The classes that will be created throughout the development can be found in Diagram 3.1. These classes ensure that the application will be able to evolve fairly easily.

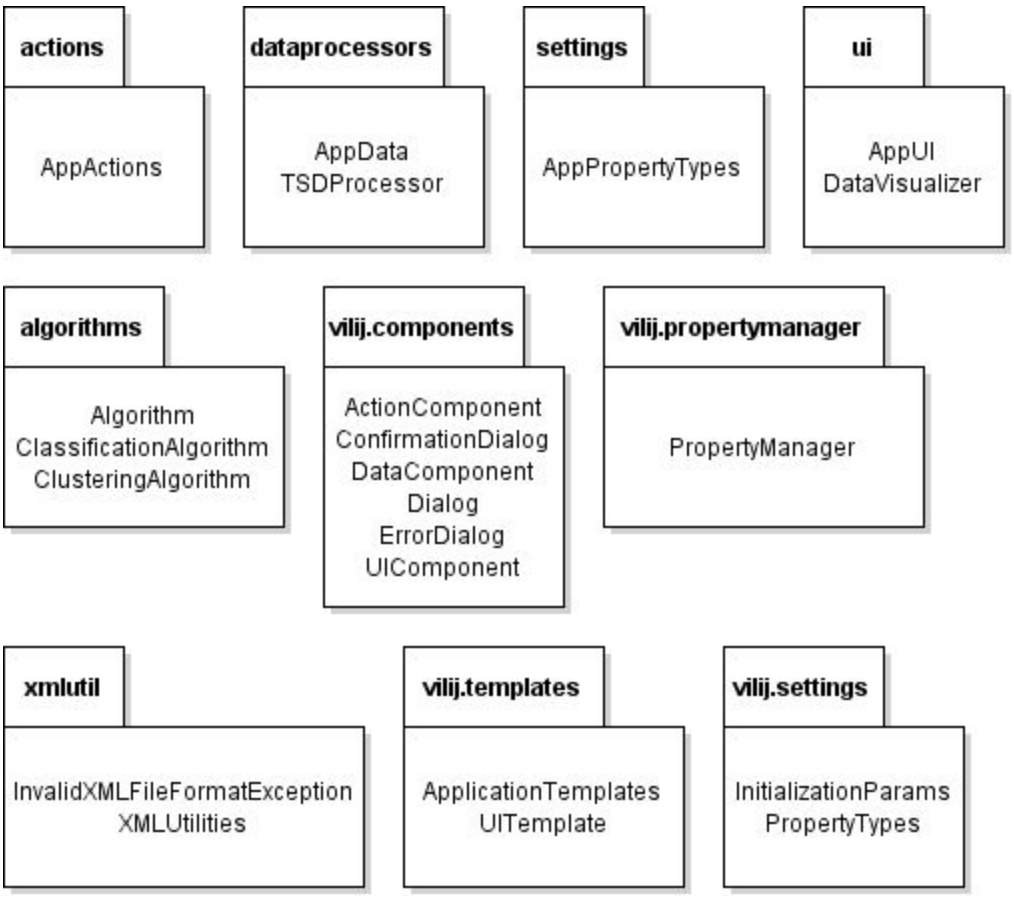



Diagram 3.1 : Package Level Software Overview Diagram

## 3.2. Java API Usage

The entirety of the software behind the Data Visualization Application will be developed in Java and will reference java classes frequently. Diagram 3.2 will list out each of the classes are used and what is being used within the class.

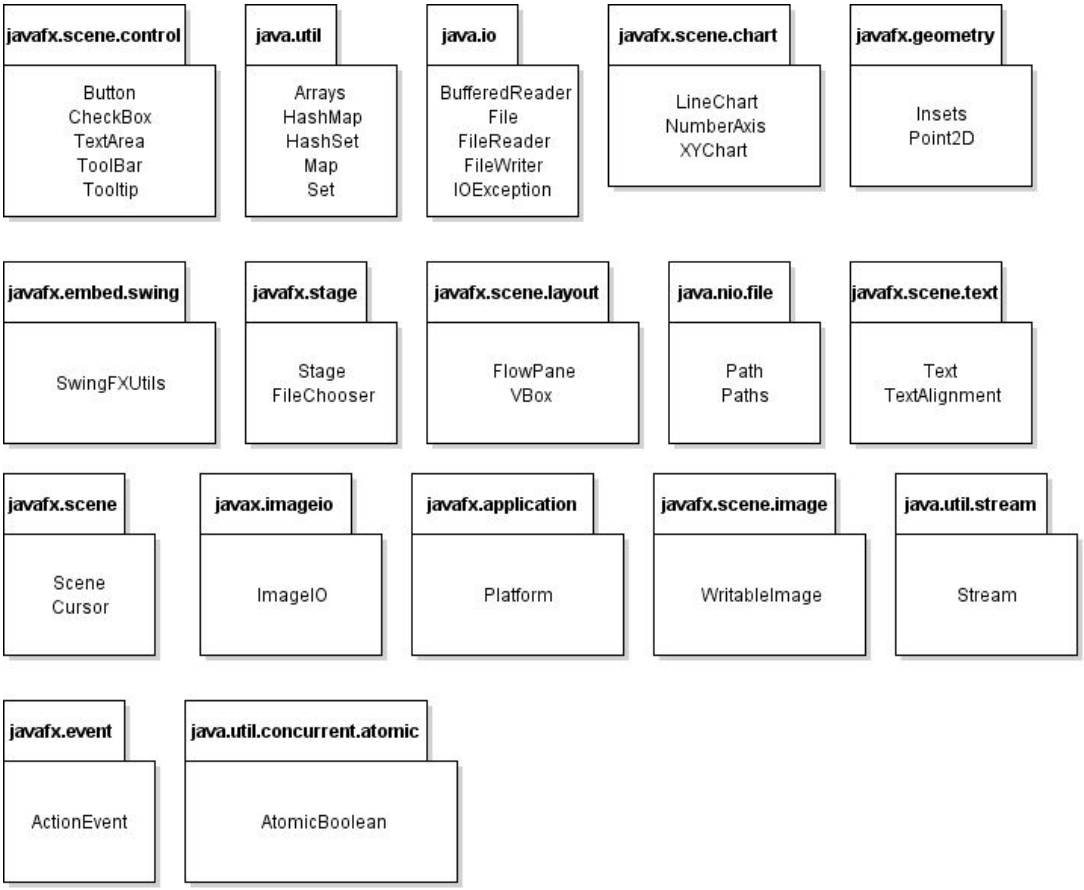



Diagram 3.2 : Java API Usages with each class that is used within the API call

## 3.3. Java API Usage descriptions

Within section 3.2, the java classes that were used throughout the project were briefly introduced. In section 3.3, the uses within the application of each section of each class will be described. The following tables, labeled 3.1-3.18 will describe the uses.

| Class | Corresponding Use |
|---|---|
| Button | For performing actions such as screenshotting the data, saving and loading the data clearing the data. |
| CheckBox | Toggle ability to edit the text area. |
| TextArea | For entering data which can be displayed. |
| ToolBar | For holding the buttons which perform various defined actions that are useful to the application. |
| Tooltip | Used to show the name of the data point when the data is hovered over. |

**Table 3.1** : This table represents the classes used in Java API's javafx.scene.control

| Class | Corresponding Use |
|---|---|
| Arrays | Used to parse the formatted data that is entered. |
| HashMap | Used to map the instances to the data points including keeping track of the labels created and the instance name. |
| HashSet | Used for traversing the current data values that were entered. |
| Map | Used for mapping each key to their value in a HashMap. |
| Set | Used for creating a set of values in a HashSet. |

**Table 3.2** : This table represents the classes used in Java API's java.util

| Class | Corresponding Use |
|---|---|
| BufferedReader | Used for reading text from a FileReader. |
| File | Used for determining the path to a specific file that will be loaded or saved. |
| FileReader | Used for reading streams of text from a specified File. |
| FileWriter | Used for writing text to a specified File. |
| IOException | Used for catching errors that may occur when reading or writing data to a File. |

**Table 3.3** : This table represents the classes used in Java API's java.io

| Class | Corresponding Use |
|---|---|
| LineChart | Used for displaying the data and drawing a line which represents the average value of the data. |
| NumberAxis | Used for creating the axes on the LineChart . |
| XYChart | Used for actually plotting the given data onto the LineChart. |

**Table 3.4** : This table represents the classes used in Java API's javafx.scene.chart

| Class | Corresponding Use |
|---|---|
| Insets | Used for formatting the workspace area. |
| Point2D | Represents the data where each Point2D is a single instance. |

**Table 3.5** : This table represents the classes used in Java API's javafx.geometry

| Class | Corresponding Use |
|---|---|
| SwingFXUtils | Used for capturing the screenshot of the data currently being displayed. |

**Table 3.6** : This table represents the classes used in Java API's javafx.embed.swing

| Class | Corresponding Use |
|---|---|
| Stage | Used for displaying the application. |
| FileChooser | Used for choosing a file to load data from or save data to. |

**Table 3.7** : This table represents the classes used in Java API's javafx.stage

| Class | Corresponding Use |
|---|---|
| FlowPane | Used for creating the layout of the workspace. |
| VBox | Used for creating a vertical layout within the workspace, next to the graph. |

**Table 3.8** : This table represents the classes used in Java API's javafx.scene.layout

| Class | Corresponding Use |
|---|---|
| Path | Used as a way to reference the current data file's path. |
| Paths | Used for creating a path to a specified directory. |

**Table 3.9** : This table represents the classes used in Java API's java.nio.file

| Class | Corresponding Use |
|---|---|
| Text | Used for creating text on the screen to provide information to the user. |
| TextAlignment | Used for aligning the text on the screen. |

**Table 3.10** : This table represents the classes used in Java API's javafx.scene.text

| Class | Corresponding Use |
|---|---|
| Scene | Used for creating the scene that the user interacts with. |
| Cursor | Used for styling the cursor when hovering over an instance. |

**Table 3.12** : This table represents the classes used in Java API's javafx.scene

| Class | Corresponding Use |
|---|---|
| ImageIO | Used for saving screenshots taken of the data. |

**Table 3.13** : This table represents the classes used in Java API's javax.imageio

| Class | Corresponding Use |
|---|---|
| Platform | Used for terminating the application. |

**Table 3.14** : This table represents the classes used in Java API's javafx.application

| Class | Corresponding Use |
|---|---|
| WritableImage | Used for creating a savable screenshot of the plotted data. |

**Table 3.15** : This table represents the classes used in Java API's java.scene.image

| Class | Corresponding Use |
|---|---|
| Stream | Used for parsing a string to determine the instances and data points within the string. |

**Table 3.16** : This table represents the classes used in Java API's java.util.stream

| Class | Corresponding Use |
|---|---|
| ActionEvent | Used for gathering details about an event that occured. |

**Table 3.17** : This table represents the classes used in Java API's javafx.event

| Class | Corresponding Use |
|---|---|
| AtomicBoolean | Used to represent whether or not an error was found within a string being parsed. |

**Table 3.18** : This table represents the classes used in Java API's java.util.concurrent.atomic

# 4. Class-level Design Viewpoint

Within the class level viewpoint, the internal structure and interaction between the various classes in the application will be described through multiple UML class diagrams.

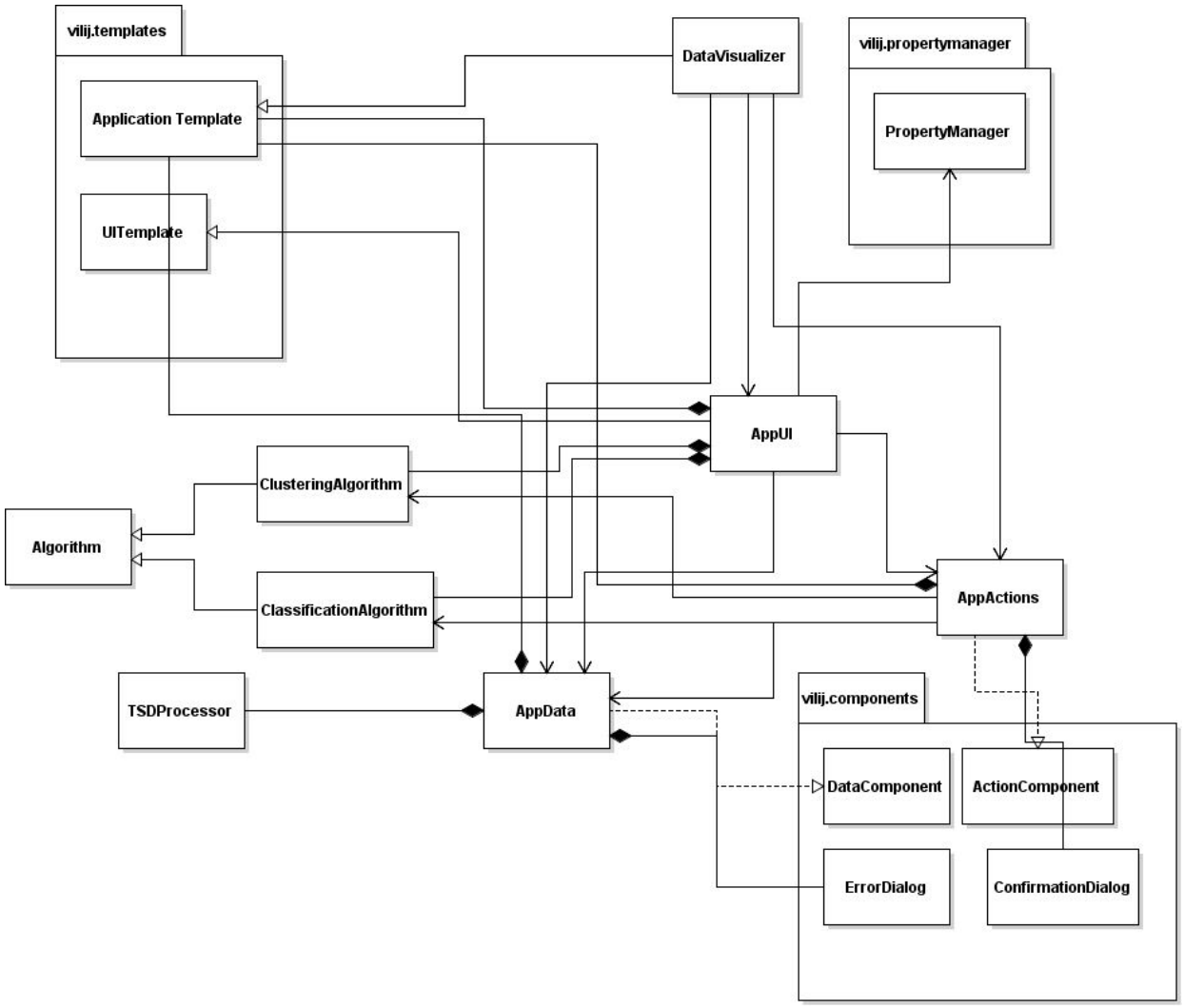



Diagram 4.1 : Data Visualization Application Class Diagram Overview

This diagram was created in order to demonstrate how each of the classes work with each other. Each of the classes have their own diagrams which include detailed interactions that occur within the class. DataVisualizer is where everything begins, it utilizes AppUI, AppActions, and AppData in order to tie everything together. From there, each component breaks down and has their own respective functionality which, when put together, creates the application and all of its functionality.

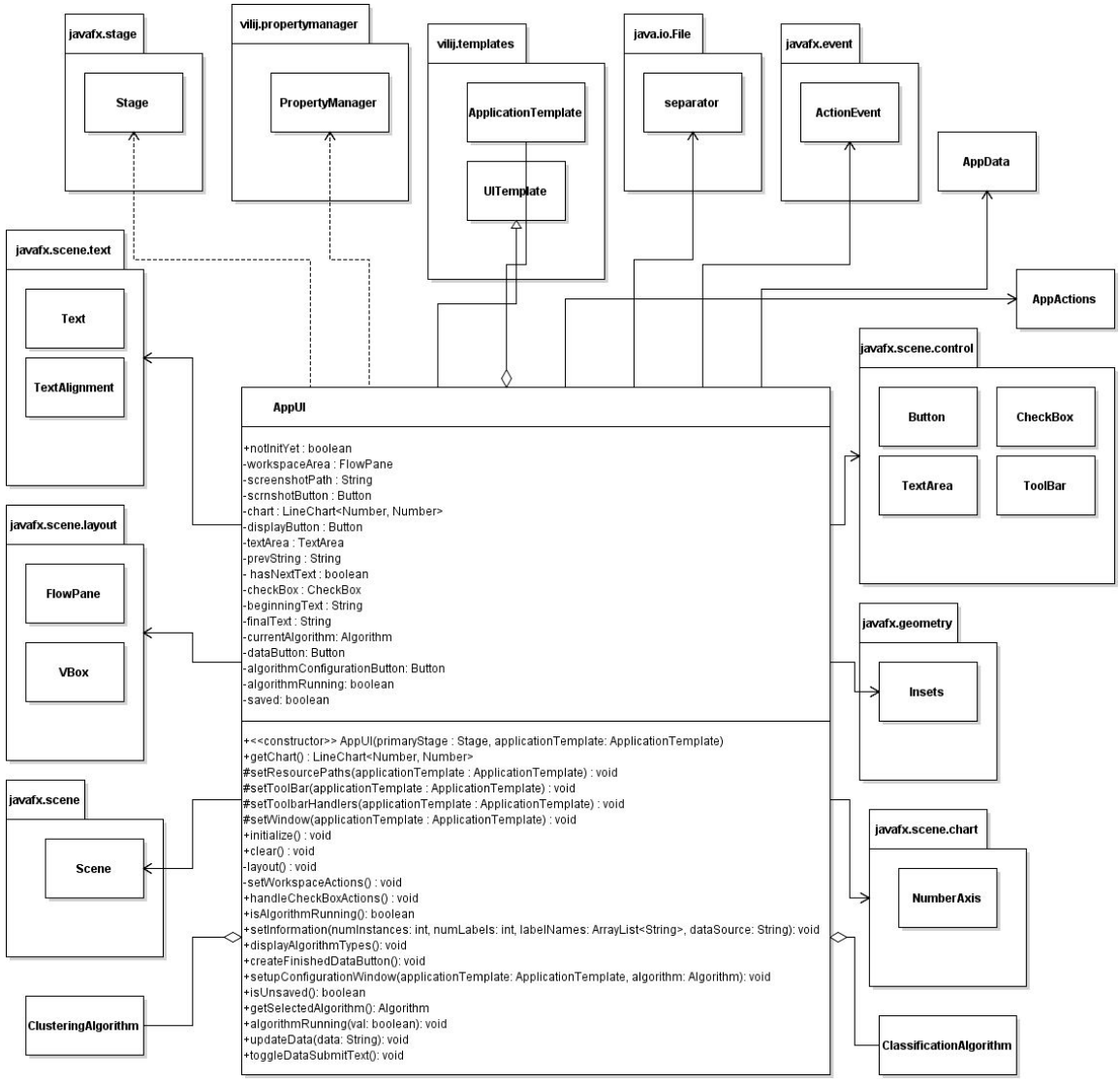



Diagram 4.2 : AppUI UML Class Diagram

The AppUI class is used to create the application and design the layout of said application and is where all the User Interface setup occurs. The AppUI class will also be used to set some of the actions that will be executed in the workspace when specified events occur such as a the text area read only mode. The AppUI class also references AppData and AppActions in order to display the data, clear the graph, and assign actions to various buttons to establish all the functionality.

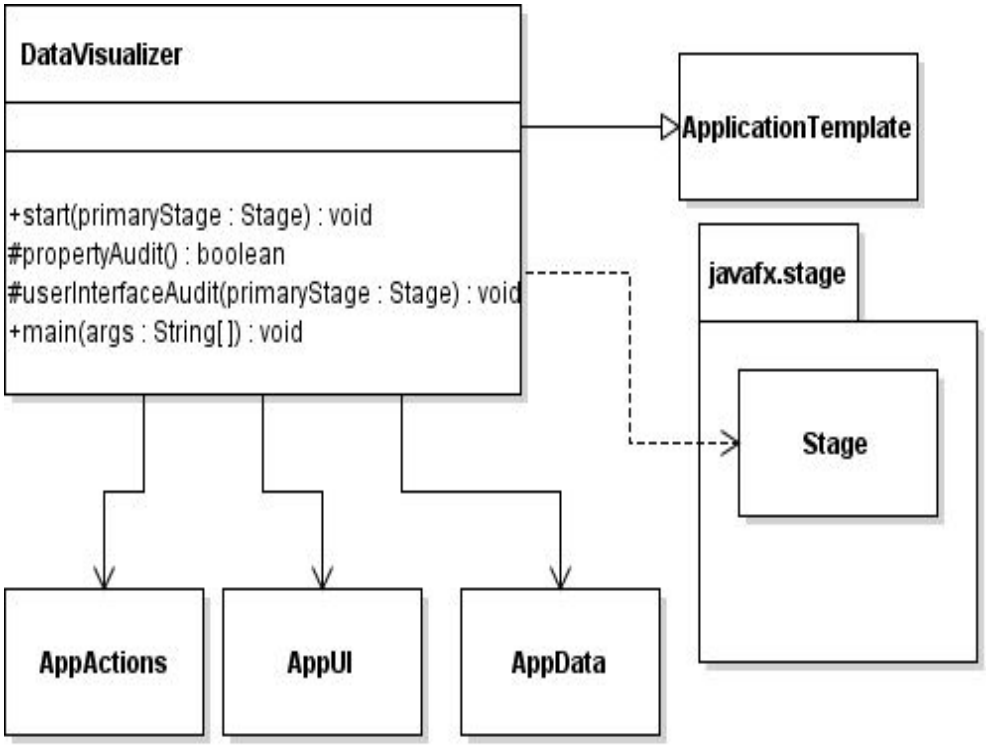



Diagram 4.3 : DataVisualizer UML Class Diagram

The DataVisualizer class is the main class in which the application runs and holds important information regarding the components of the application as well as the XML properties. This class will set up the Action, Data and User Interface components of the application which will then set up the application by working together to achieve the required functionality. Data Visualizer is the initial class which is called when running the application.

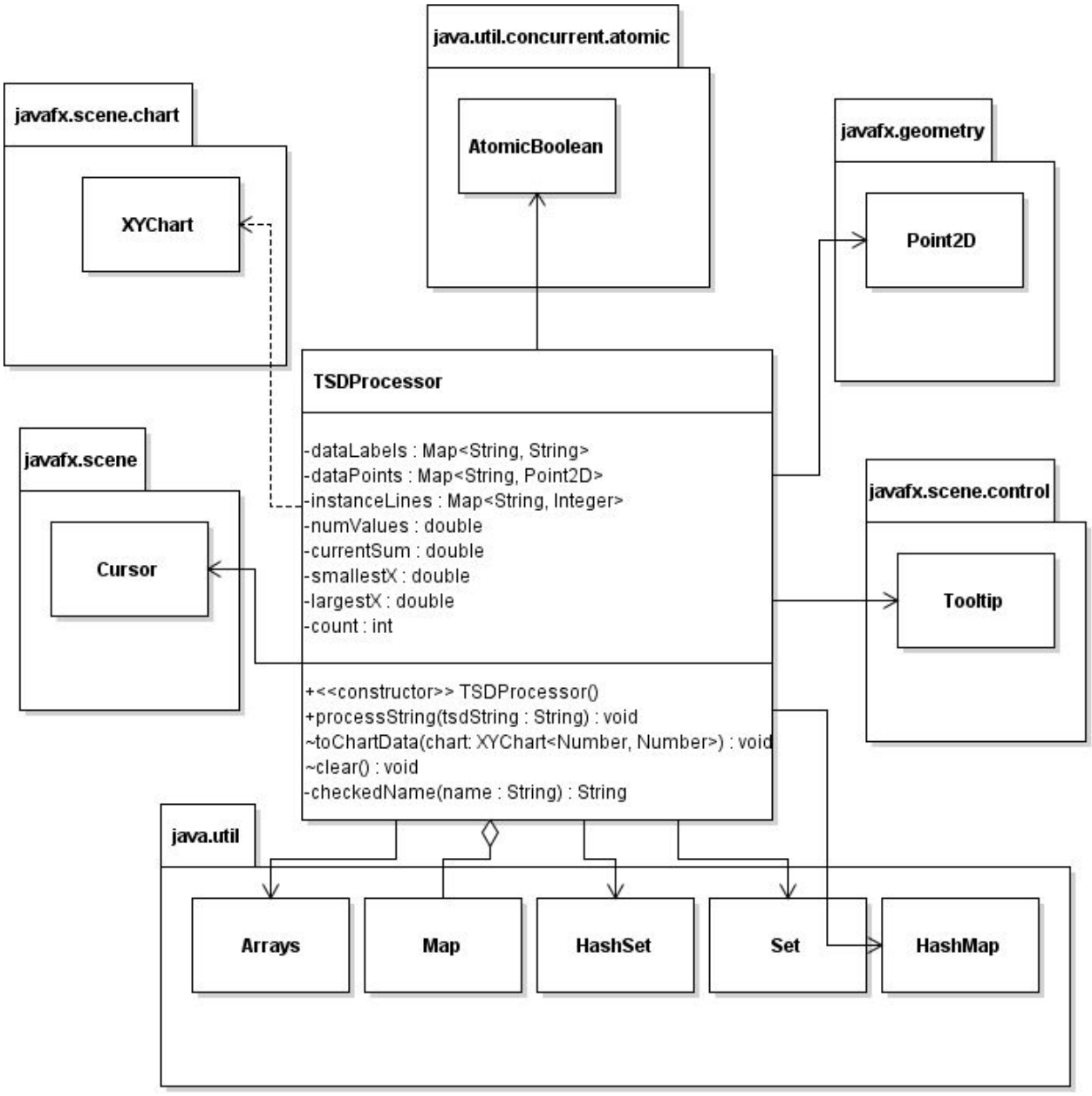



Diagram 4.4 : TSDProcessor UML Class Diagram

The TSDProcessor class has a purpose of parsing data which is separated by indents. TSD stands for Tab-Separated-Data and is a format that we use to determine the data values entered. The responsibility of this class is to process the string of data given, saving the instances and values of the data, alongside the average value of the data points within the object. At a later point in time, the TSDProcessor will then graph the processed data on a graph that is passed to the processor and add a Tooltip to each data entry. The TSDProcessor can also clear the graph.

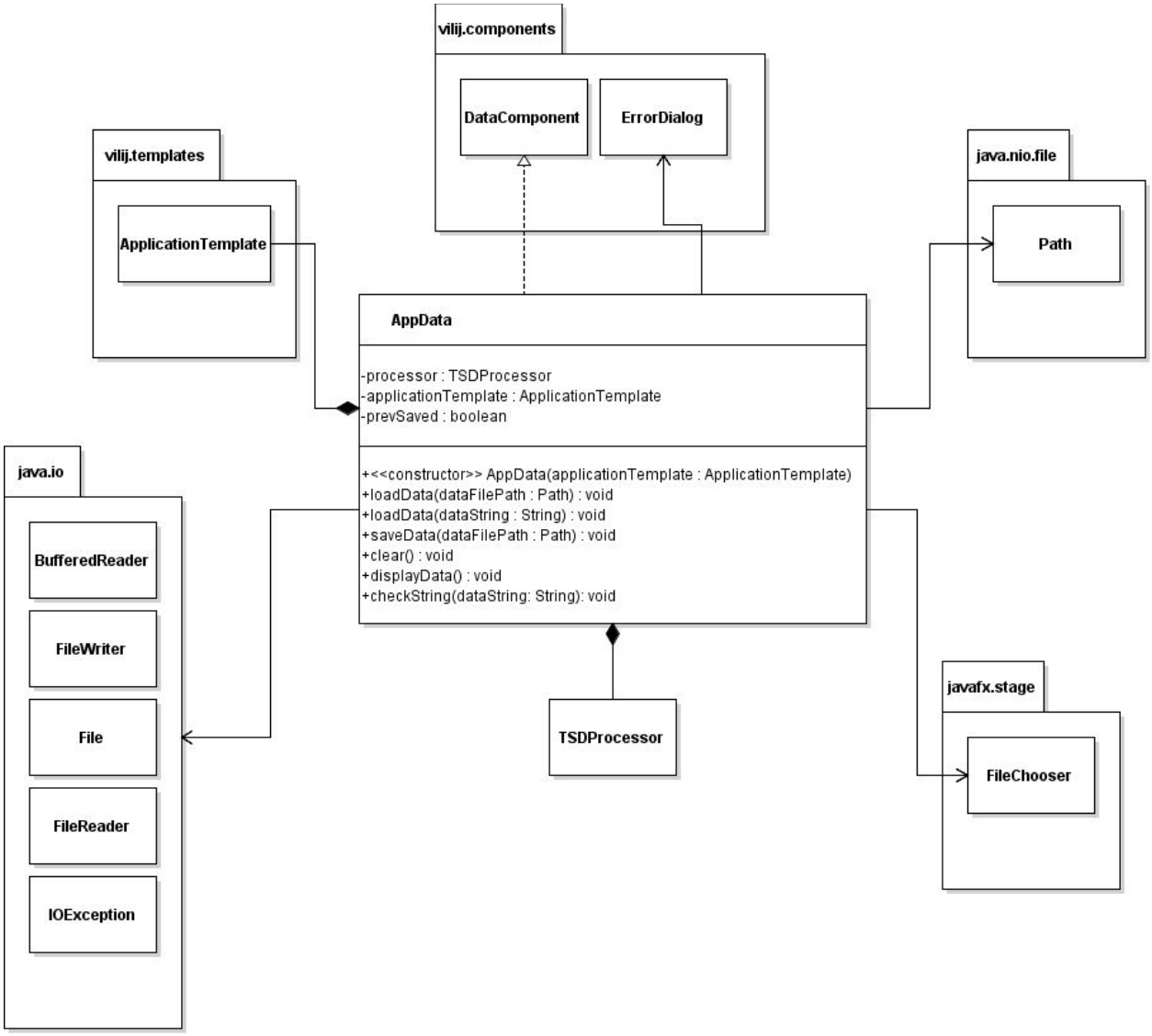



Diagram 4.5 : AppData UML Class Diagram

The AppData class will serve as the data component of the project. This class has the ability to create a processor which will parse the strings in the data fields and return data to the User Interface. The important part of this class comes from its ability to check given data and determine whether or not it is valid and act accordingly. The AppData class can also save the data in the processor if it is given a desired path, and load data from either a specified file or a given string. Finally, the AppData class also has the responsibility of updating the graph in the user interface.

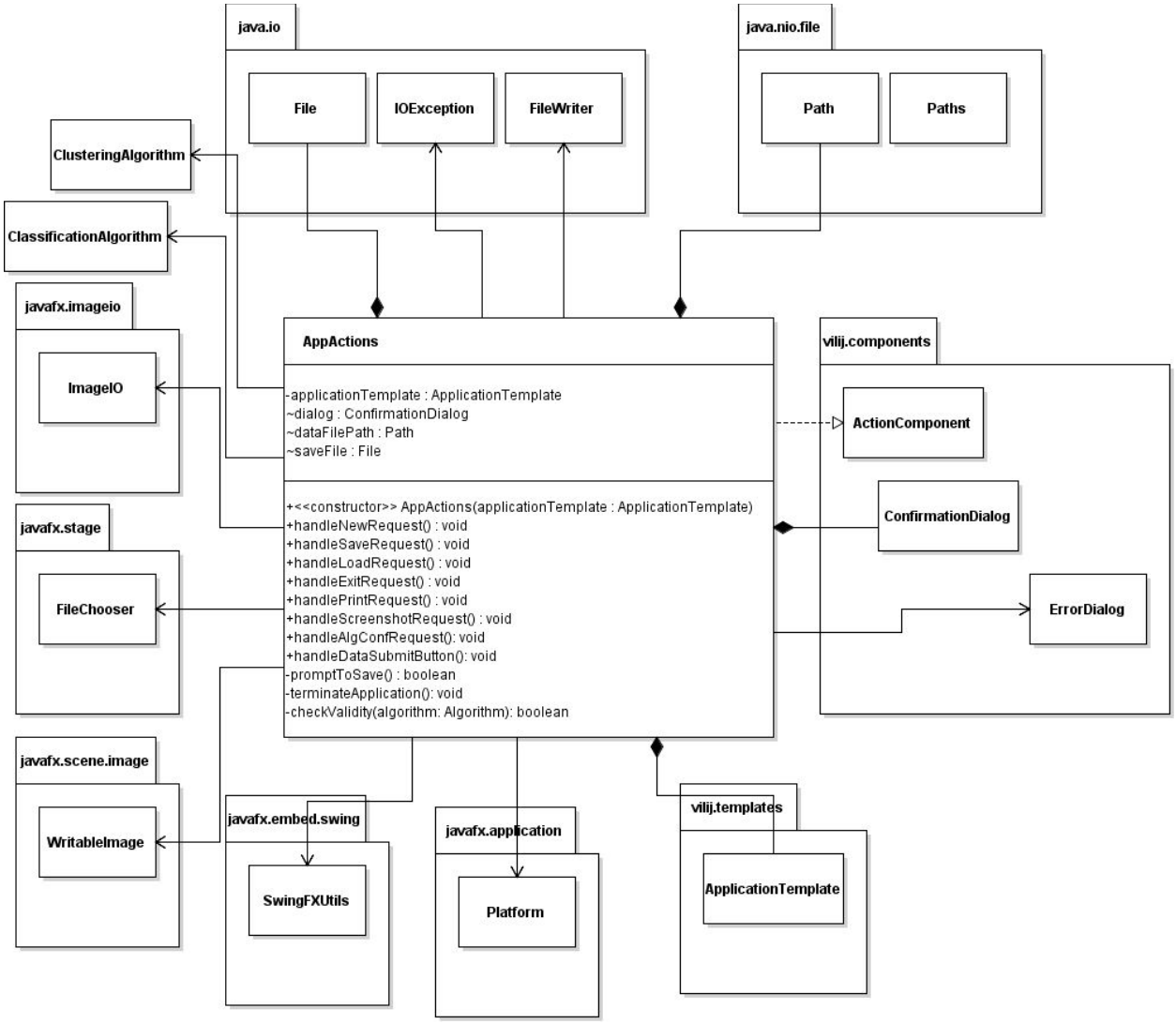



Diagram 4.6 : AppActions UML Class Diagram

The AppActions class in the application deals with events that may occur, such as the user wanting to create new data, take a screenshot, load data from a specified file, etc. Alongside this functionality, the AppActions class can also create FileChoosers for the user to pick a file to save data to or load from. Another use of the AppActions class is storing important information regarding what the user may want to occur. An example of this is storing the file that was previously saved to and when the user then clicks save, the data in the file is automatically updated without the user having to reselect the file. The final important method that this class contains is the terminateApplication method which will simply end the application when the user clicks the exit button and selects the correct option.

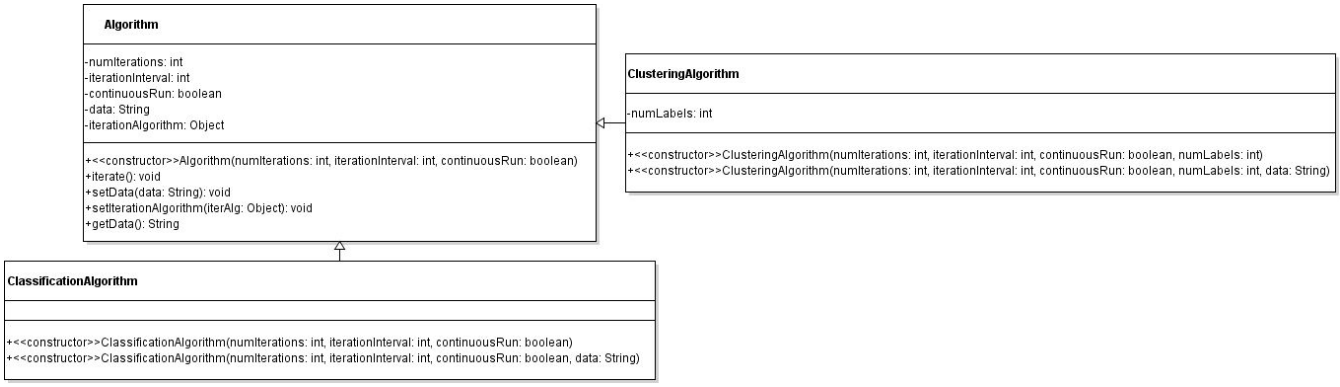



Diagram 4.7 : Algorithm, ClusteringAlgorithm, and ClassificationAlgorithm UML Class Diagrams

The Algorithm class is a superclass to ClassificationAlgorithm and ClassificationAlgorithm simply because they have similar but not equal attributes. ClusteringAlgorithm has an additional field called numLabels. The two subclasses hold specific information regarding the algorithms that will be used and can be configured which is demonstrated in Use Case 7. The algorithms will be running in a separate thread than the gui in order to increase speed and efficiency; the algorithms will require large amounts of time and for that reason it is important to multithread the application. Each algorithm can have a specified way to iterate over the data, changing the data each iteration. The algorithm will never change the original data set which is useful for reusing data. In order to load data into the algorithm, simply use the setData method provided and to iterate over the data, call the iterate method then get the new data by calling the getData method on the algorithm.

# 5. Method-level Design Viewpoint

Within the method-level design section, the sequence diagrams of the Use Cases within the application will be described. There are 10 separate use cases that will take place although some of the use cases have multiple scenarios. If a use case has multiple scenarios, each scenario will be differentiated by indicative numbers under the sequence diagram.

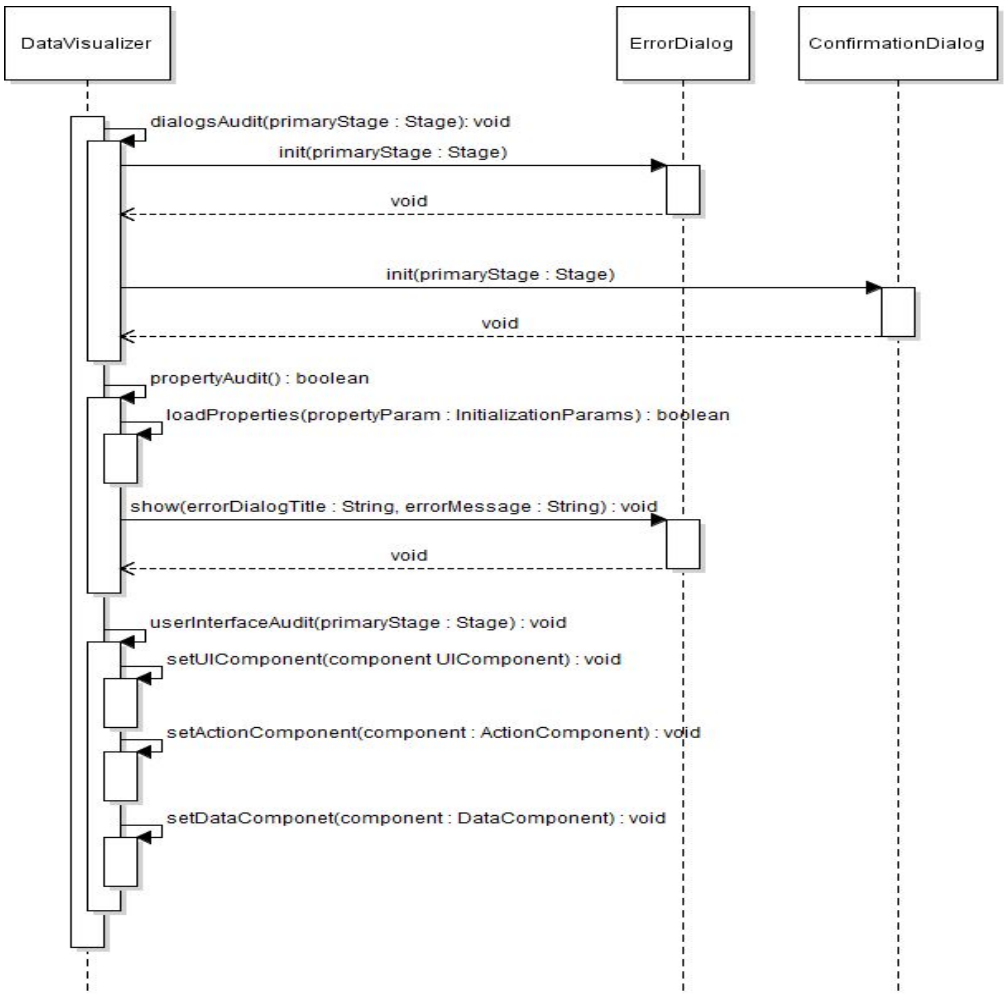



Diagram 5.1 : Start Application Use UML Sequence Diagram(Use Case 1)

The entirety of the application begins in DataVisualizer when start method is overridden, calling dialogsAudit, propertyAudit, and userInterfaceAudit which load all the properties of the application, generate the user interface, data component, and action component. From here, the interactions between the user and the application become more complex.

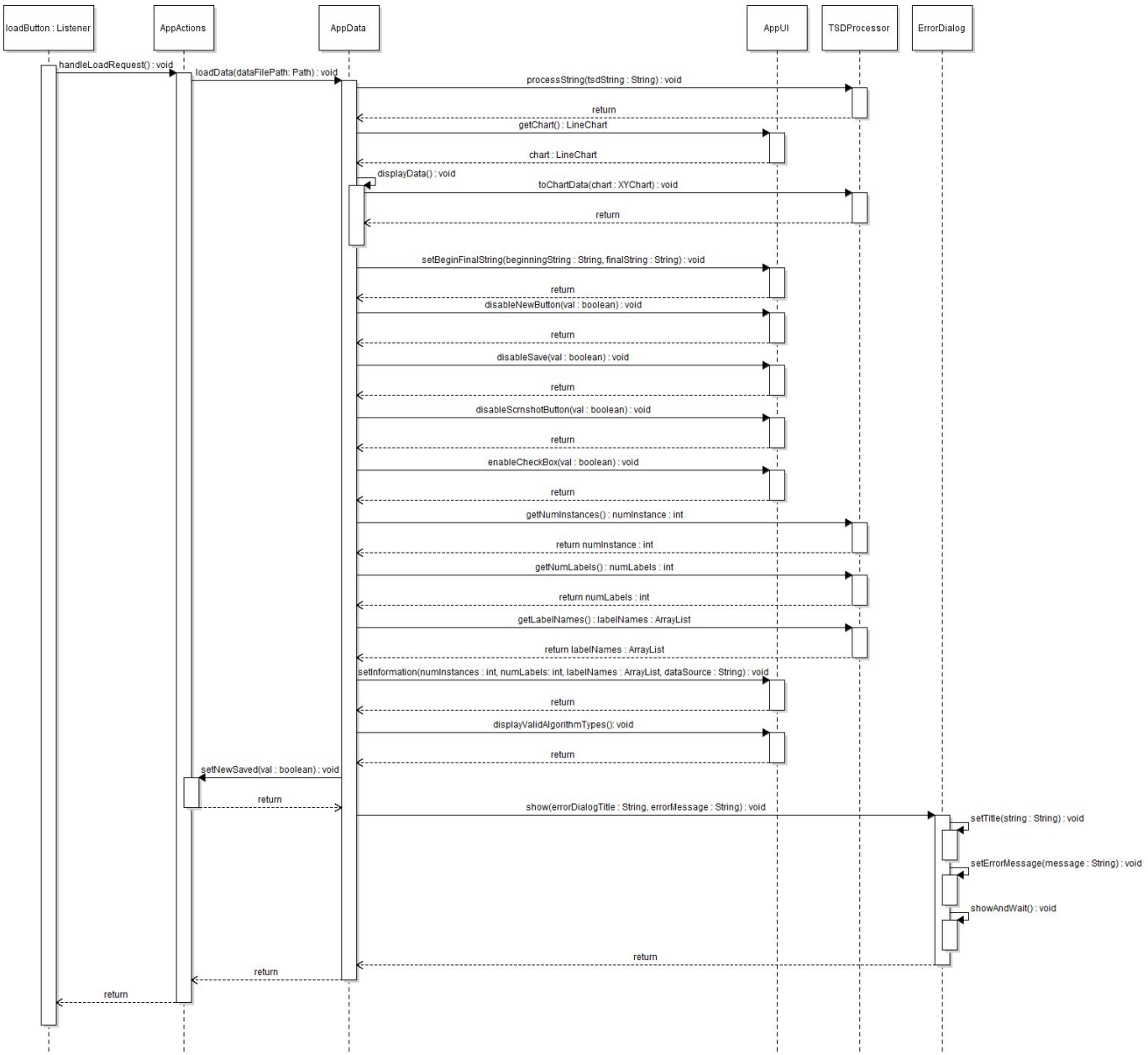



Diagram 5.2 : Load Data UML Sequence Diagram(Use Case 2)

This sequence diagram deals with a user loading data from a file they choose. When the loadButton is clicked, a filechooser will appear and allow the user to select the file they would like to load. From there, the AppData class will take the Path, load the data and process it using the TSDProcessor class. The AppData class will communicate with TSDProcessor and pass along information regarding the data to the user interface, displaying the data on the graph and displaying the valid algorithm types based off the data loaded.

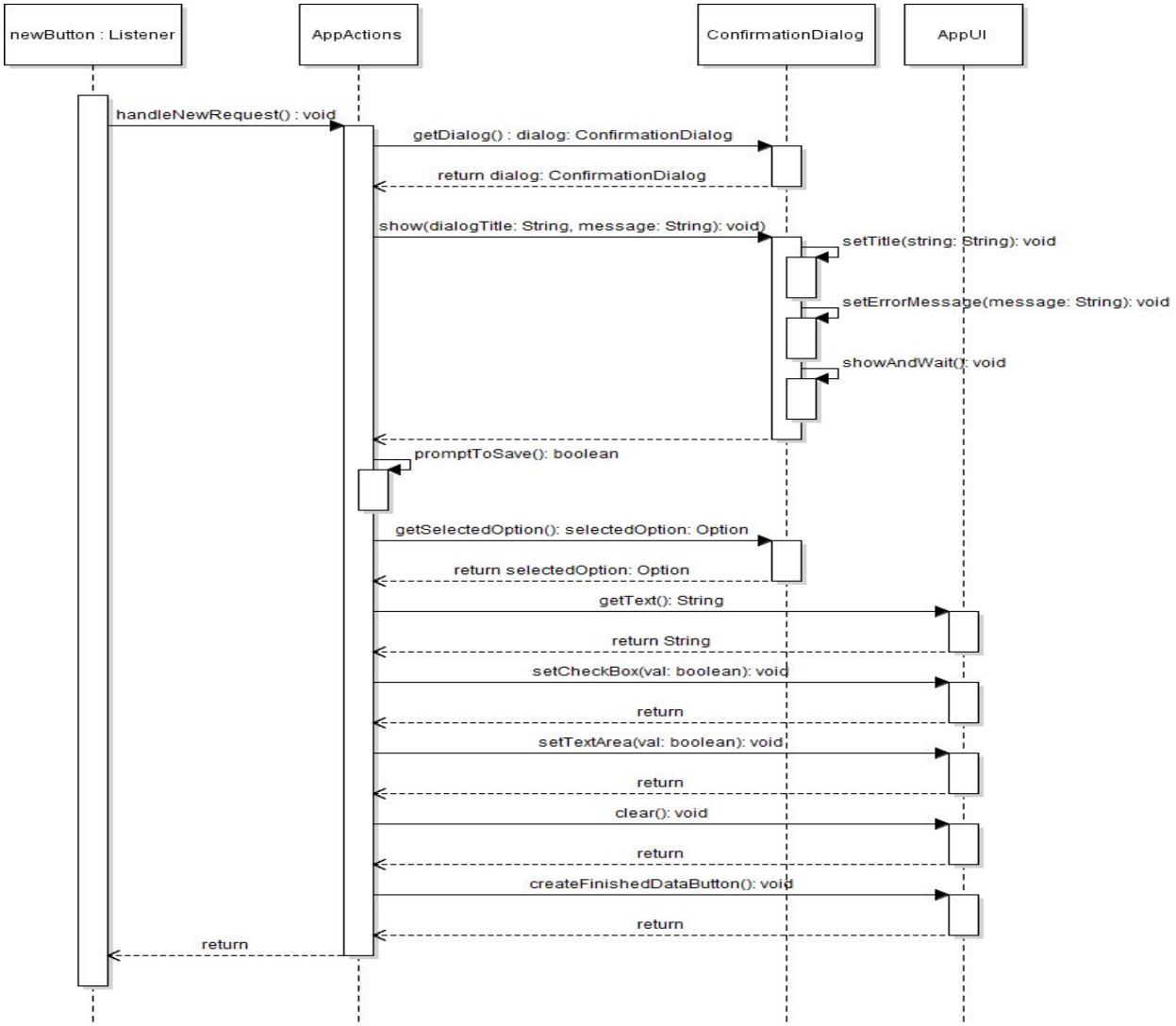



Diagram 5.3 : Create New Data UML Sequence Diagram(Use Case 3)

The sequence diagram found in diagram 5.3 is showing what occurs when a user clicks the new button. Initially, a dialog pops up asking the user to save their data, this sequence diagram represents a user responding yes to the dialog. It will save the data, then will enable the text area, uncheck the read-only checkbox and clear the text area before it creates a finishedDataButton. The following diagram, Diagram 5.4, will show the final part of this use case and what happens when the finishedDataButton is clicked.

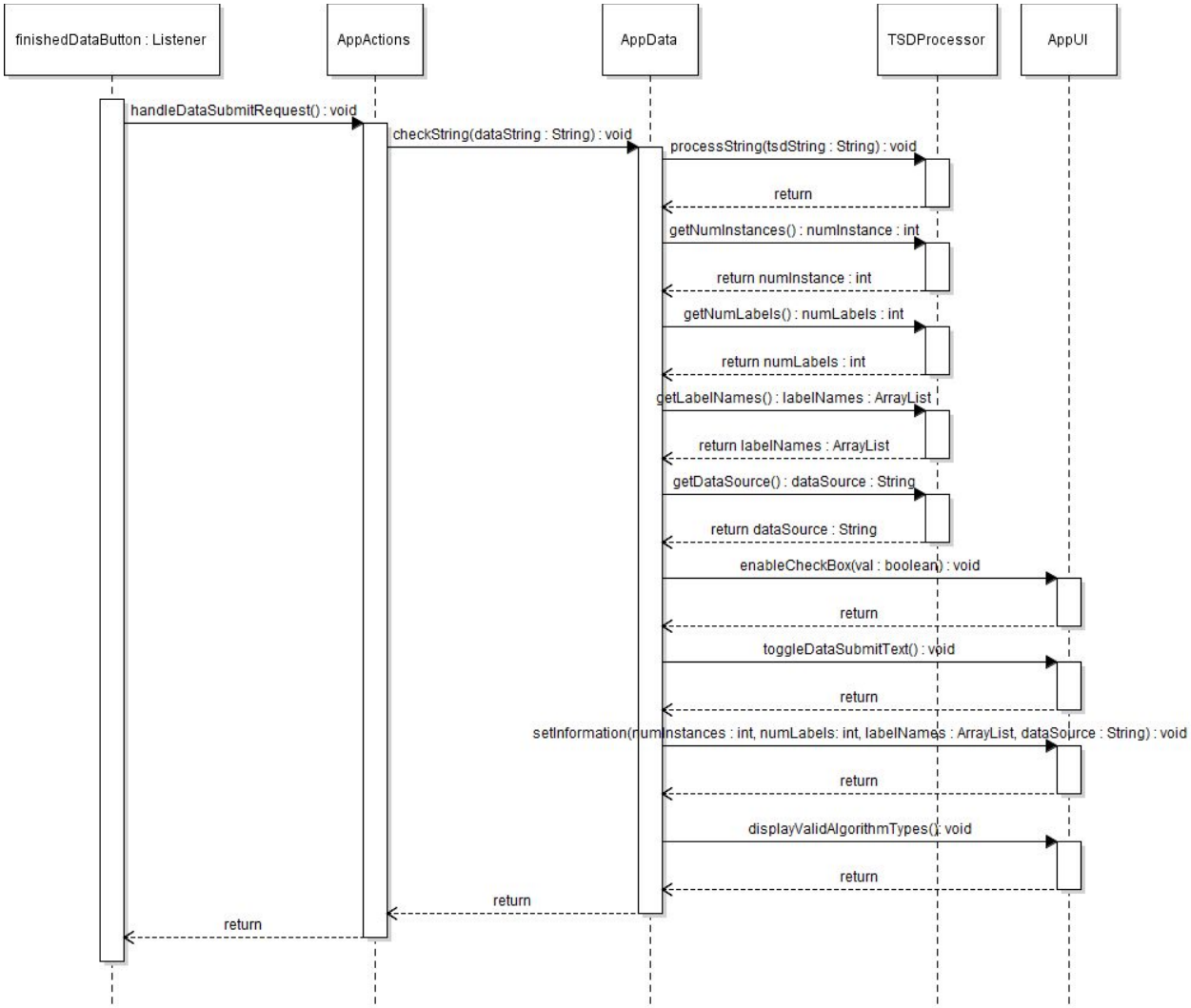



Diagram 5.4 : Finished Data Entry Button Response UML Sequence Diagram

This diagram is useful for the last part of use case 3. When the user is finished entering their data and they click the finishedDataButton, the application will take the data they entered and validate the data by attempting to process it in TSDProcessor. Assuming the data is valid, the AppData class will then enable the read-only checkbox and disable the textarea in AppUI. Finally, the UI will update the information of the data on the main window along with the valid algorithms that can be used for this data set.

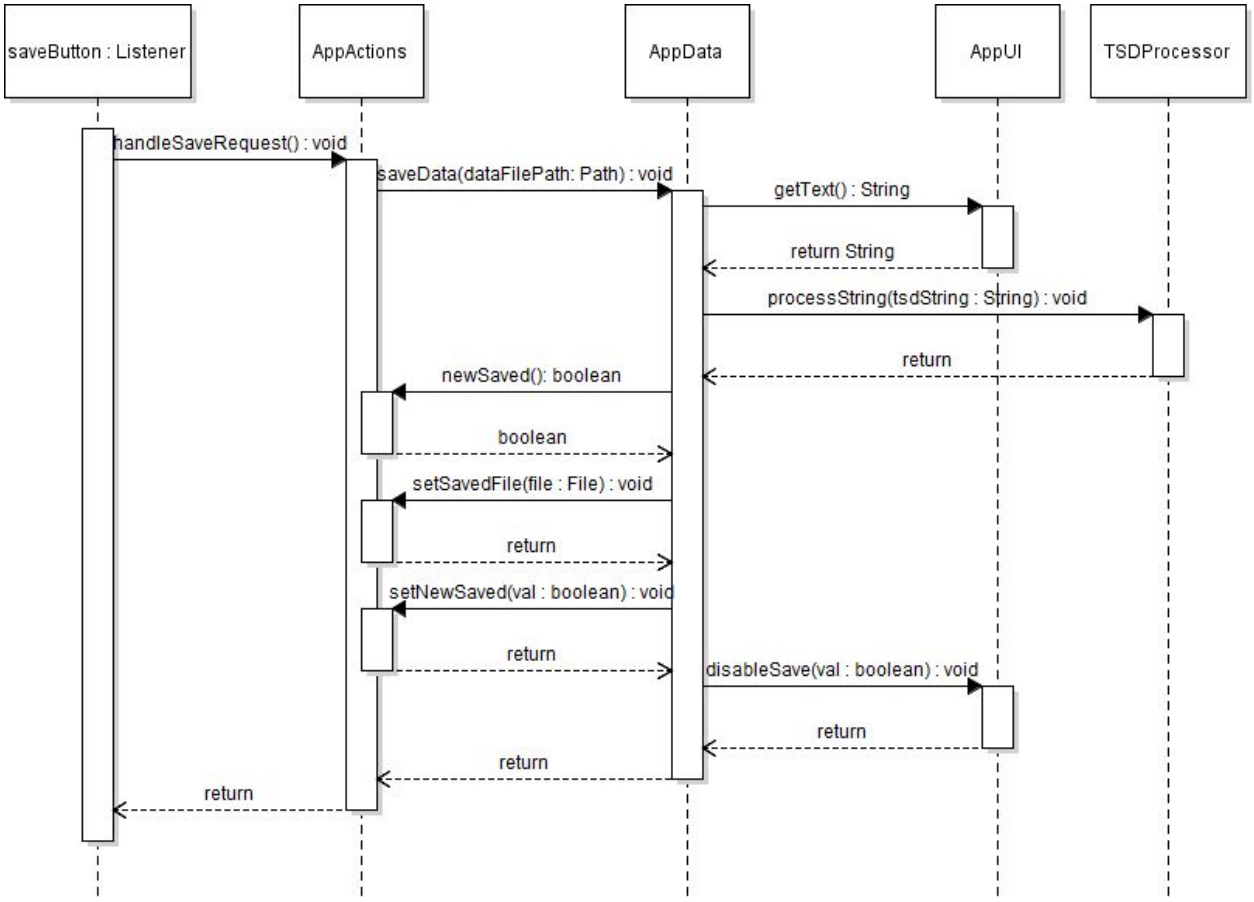



Diagram 5.5 : Save Data UML Sequence Diagram(Use Case 4)

The sequence diagram found above shows the process that occurs when the user attempts to save data in the application. Once the user presses the saveButton, the listener calls the handleSaveRequest method inside of AppActions which opens a FileChooser for the user to determine where to save the file. After this, the file path is passed to AppData through the saveData method which grabs the current data in the application from AppUI and checks if the data is error free. After confirming that the data is valid, a few variables will be configured to help the application to work as efficiently as possible before returning.

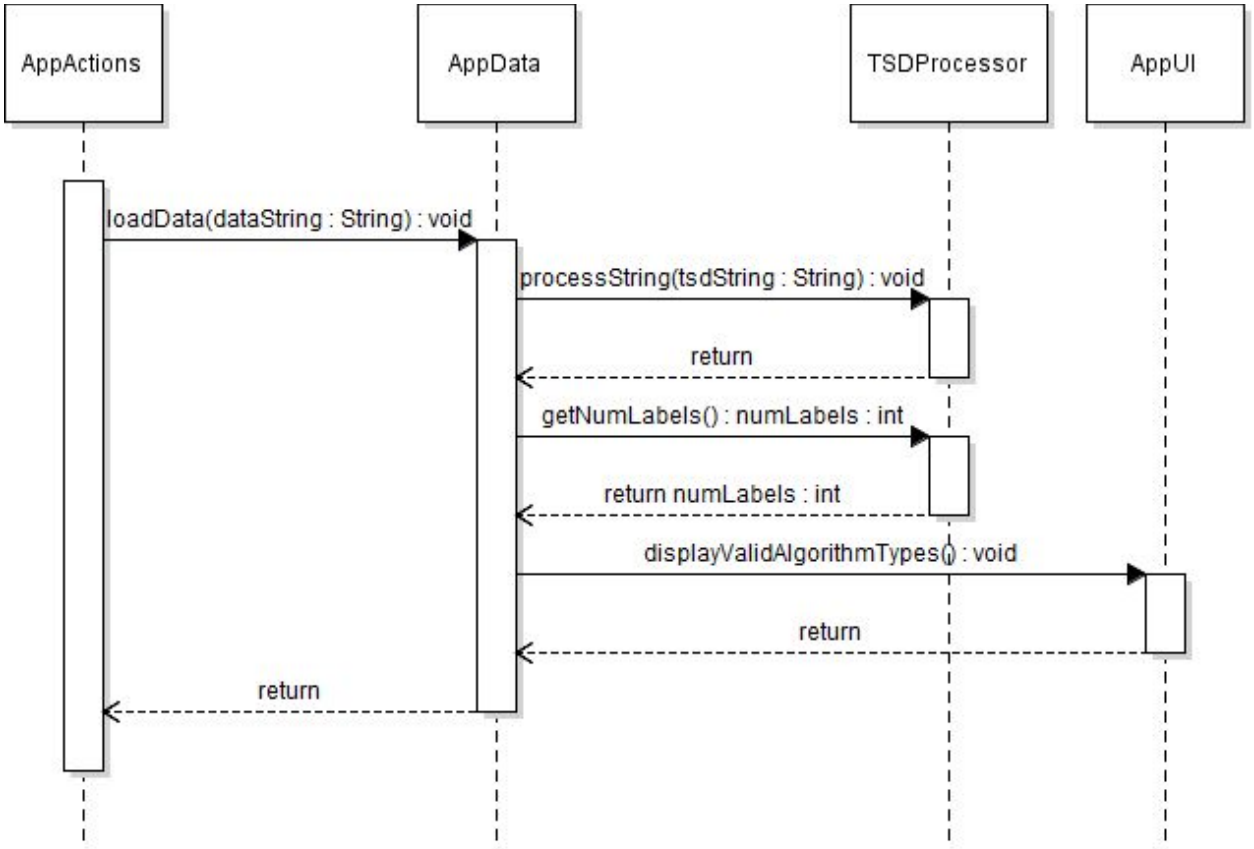



Diagram 5.6 : Select Algorithm Type UML Sequence Diagram(Use Case 5)

This particular sequence diagram deals with what happens after a user loads data into the application by manually typing it into the text area. After the user chooses to load the data, the application should display the algorithm types that are available for the data set. Therefore, before the displayingValidAlgorithmTypes method is called, the application must check the number of labels to make sure classification is able to be used, if not, the application will only display clustering.

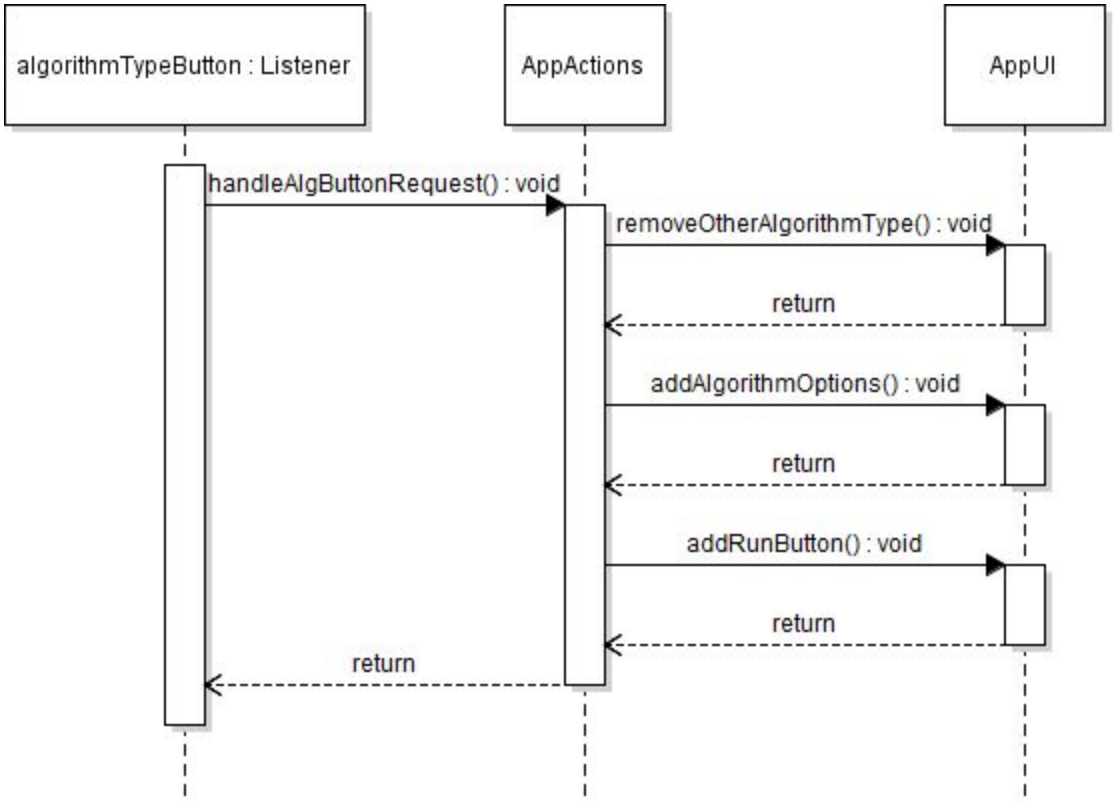



Diagram 5.7 : Select Algorithm UML Sequence Diagram(Use Case 6)

After the user selects the algorithm type, the application should display the options that are available for the algorithm type that they chose and remove the other algorithm type that they did not choose. The application can do this by calling the removeOtherAlgorithmType method on AppUI which will determine which algorithmType was picked by the user and then the application will app the corresponding algorithm options, including the configuration button, for the selected algorithm type just before it adds the run button(which will be implemented in Use Case 8) and returns.

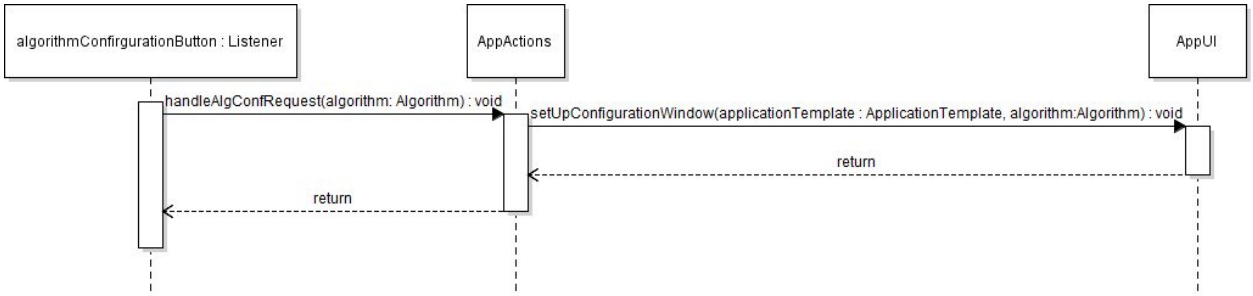



Diagram 5.8 : Select Algorithm Running Configuration UML Sequence Diagram(Use Case 7)

This sequence diagram is fairly simple; once the algorithmConfigurationButton is pressed on any of the algorithm buttons, the listener will pass along the algorithm that the configuration button was binded to to AppActions through the handleAlgConfRequest method. After that, AppActions will call AppUI and make a new scene that pops up the configuration window for the algorithm. In that window, the specifications for the algorithm can be selected and adjusted which will change them in the algorithm object. The different algorithm types will have a few different parameters to adjust but will be similar for the most part.

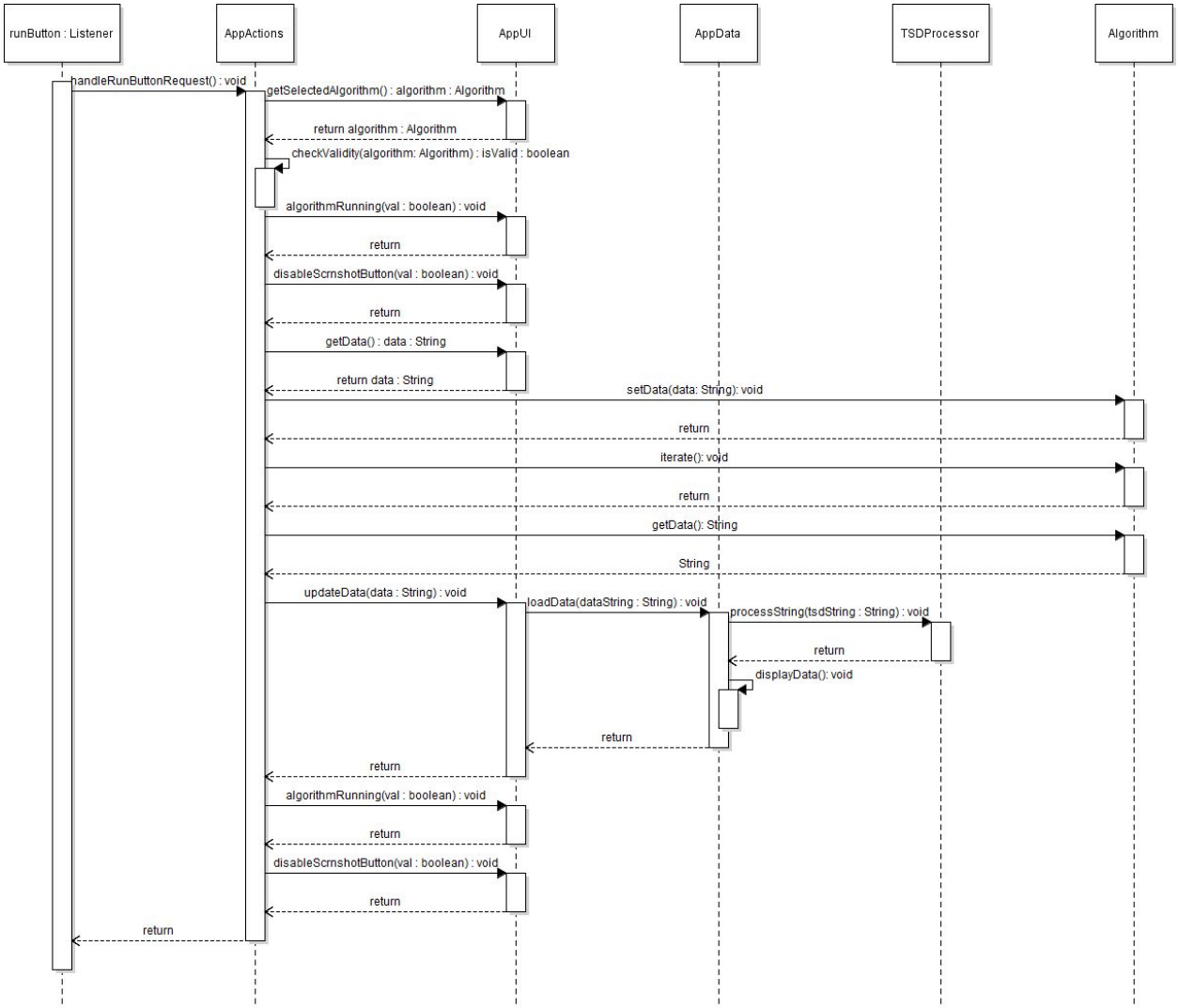



Diagram 5.9 : Running an Algorithm UML Sequence Diagram(Use Case 8)

The sequence diagram for Use Case 8 is as displayed above. The diagram shows that once the runButton is clicked, the listener will call the handleRunButtonRequest method in AppActions which will try to validate that the selected algorithm has all the required parameters fulfilled through the configuration. Once the algorithm is validated, AppActions will then pull the data from the AppUI and feed it to the Algorithm which will then iterate and then return the updated data only to update the graph. If the user had selected continuous run, the algorithm will continue to iterate through the data set until the end condition. During the time the algorithm is running, the screenshot button will become disabled. If the algorithm was not set to continuously run, the user will have to press run whenever he would like to see the progression of the algorithm.

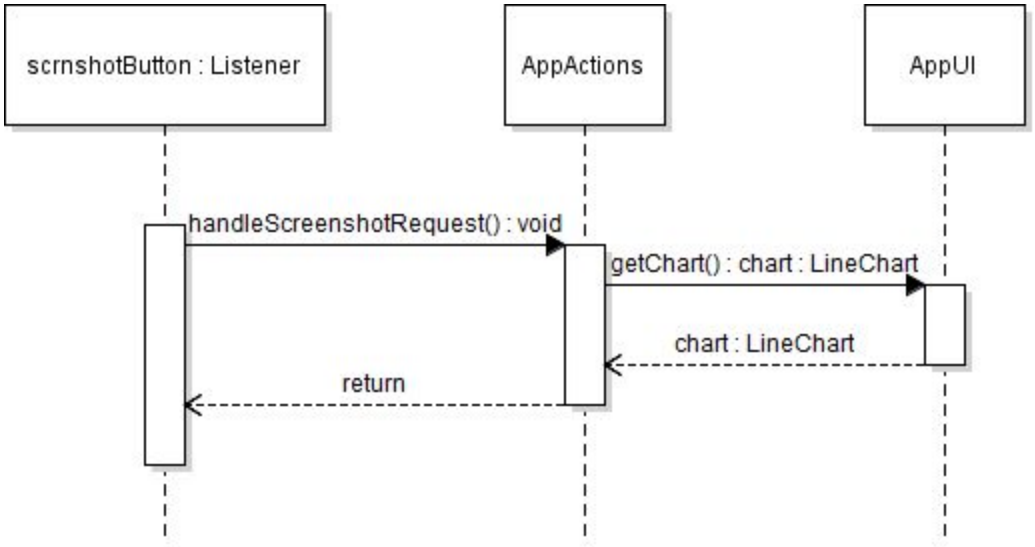



Diagram 5.10 : Export Data Visualization as an Image UML Sequence Diagram(Use Case 9)

This diagram describes the process behind taking screen captures of the graph. The chart has a method called snapshot which allows you to capture the graph and only the graph. Once the screenshot button is clicked, the listener will call the handleScreenshotRequest in AppActions which will proceed to get the chart. Once the chart is obtained, you can use the snapshot method to capture the chart. Finally, save the image into the resources directory.

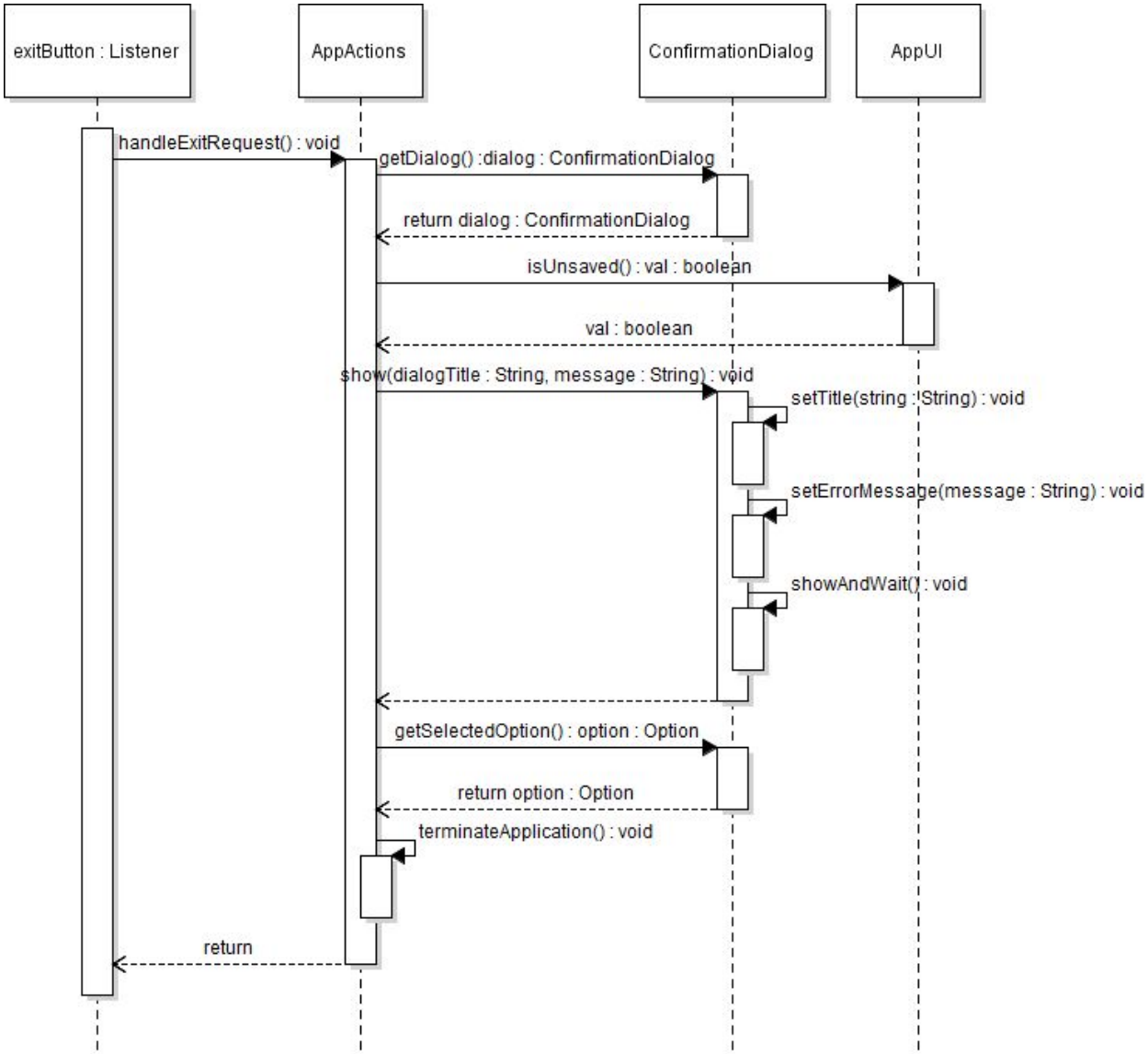



Diagram 5.11 : Exit Application(Use Case 10)

This is the final use case, although it is separated into five separate scenarios. Diagram 5.11 describes the scenario where there is unsaved data within the application and the user wants to exit the application. This diagram specifically covers the scenario where the user doesn't care to save the data and just wants to exit the application, losing the data. The exitButton has a listener which calls handleExitRequest in AppActions, prompting a dialog to pop up, asking if the user wants to save his data. Since we are assuming the user doesn't want to save the data, we just simply terminate the application by calling Platform.exit().

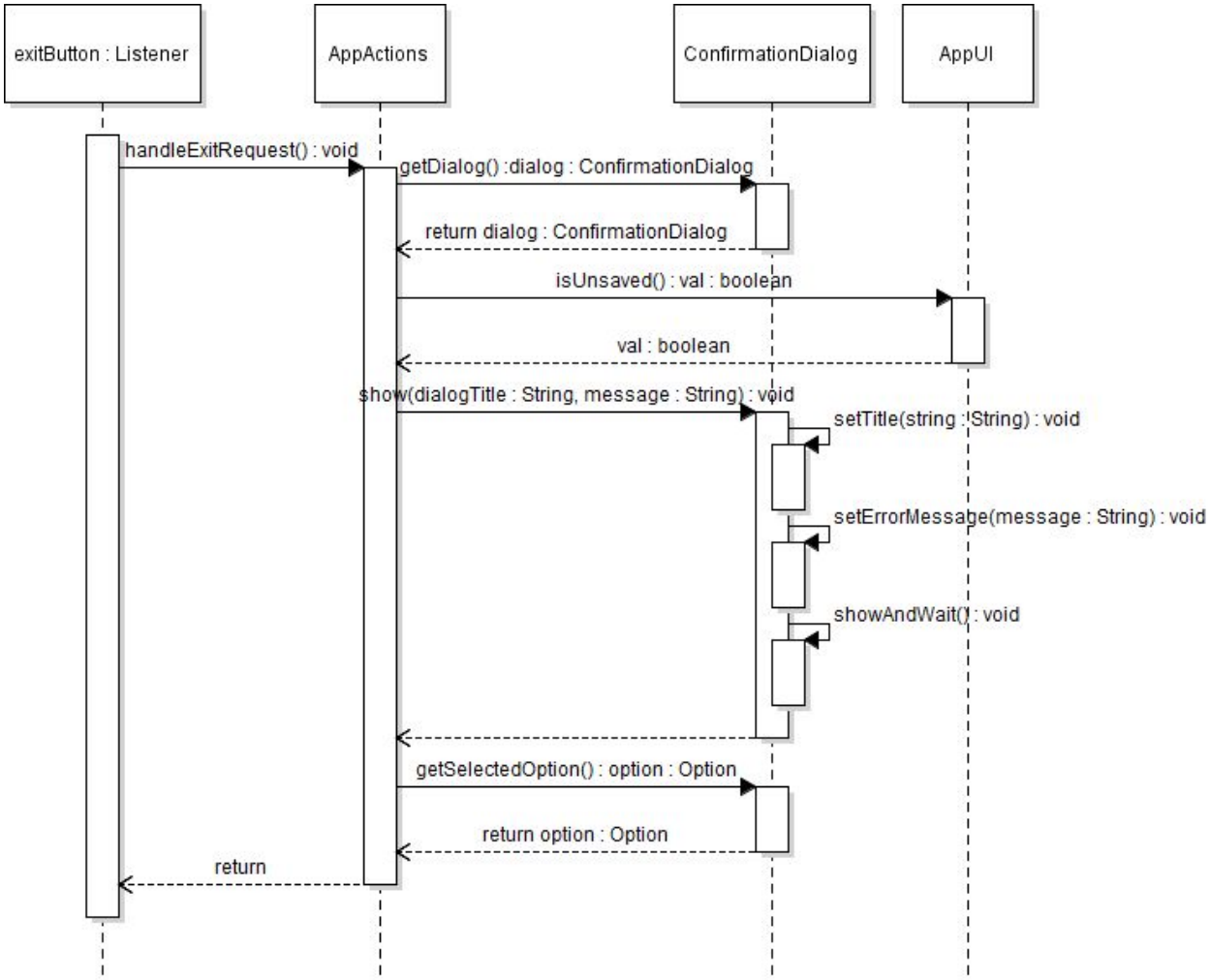



Diagram 5.12 : Exit Application(Use Case 10)

Diagram 5.12 covers what happens when the user doesn't want to close the application and would rather return to the application. The same process that occured in the Diagram 5.11 occurs now, although instead of exiting, the application just returns. No data is saved, the application just continues running like it previously was.

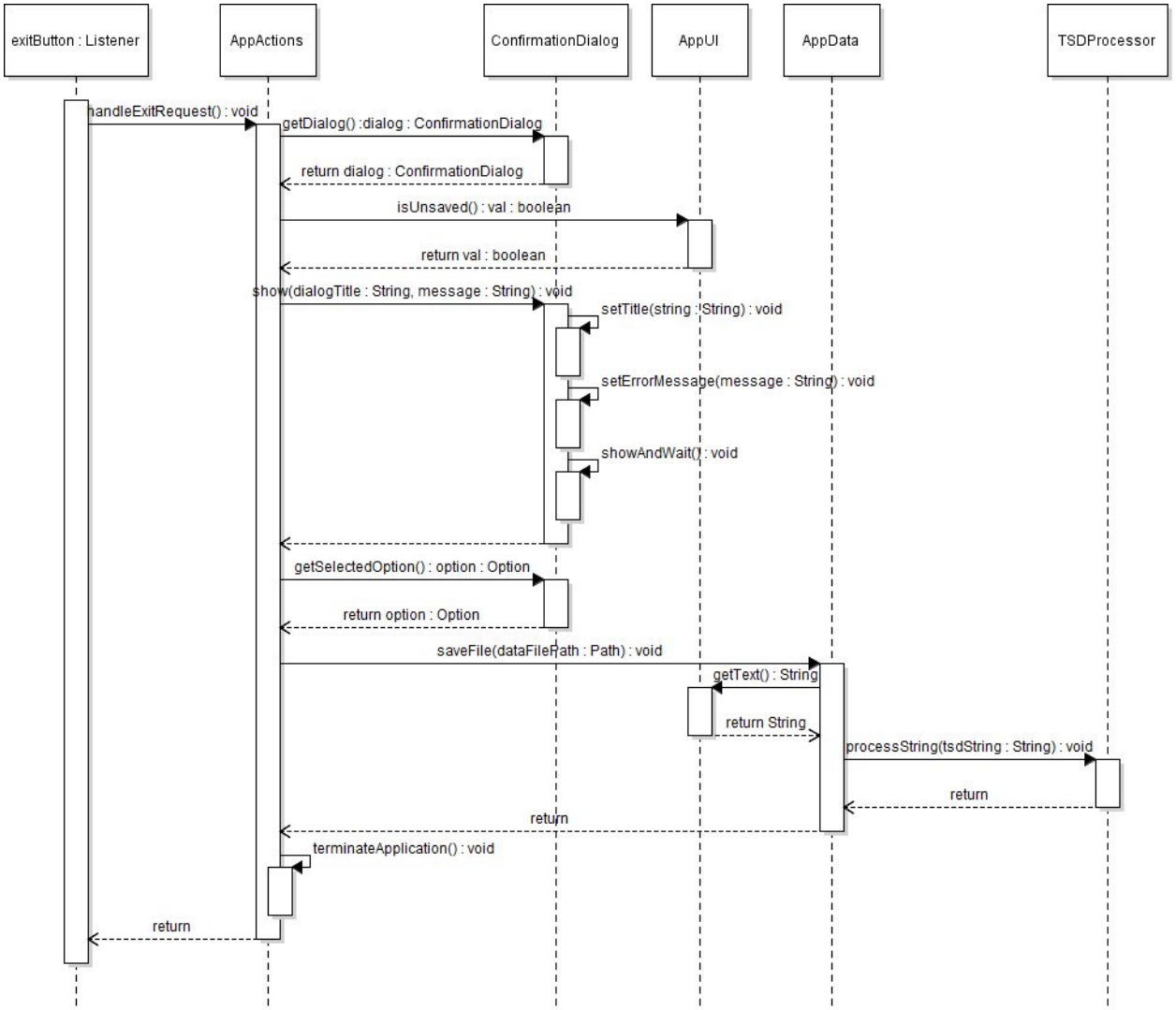



Diagram 5.13 : Exit Application(Use Case 10)

This is the final sequence diagram that deals with exiting the application with unsaved data. Following the same path as the past two diagrams, the exitButton has a listener which calls handleExitRequest in AppActions, prompting a dialog to pop up, asking if the user wants to save his data. Now we are assuming the user wants to save the data, then exit the application. We can do this by prompting the user to select a file to save to using FileChooser, then call the saveFile method in AppData to save the data to the specified file. Now that the data is saved, it is safe to exit the application using the previously defined terminateApplication method in AppActions.

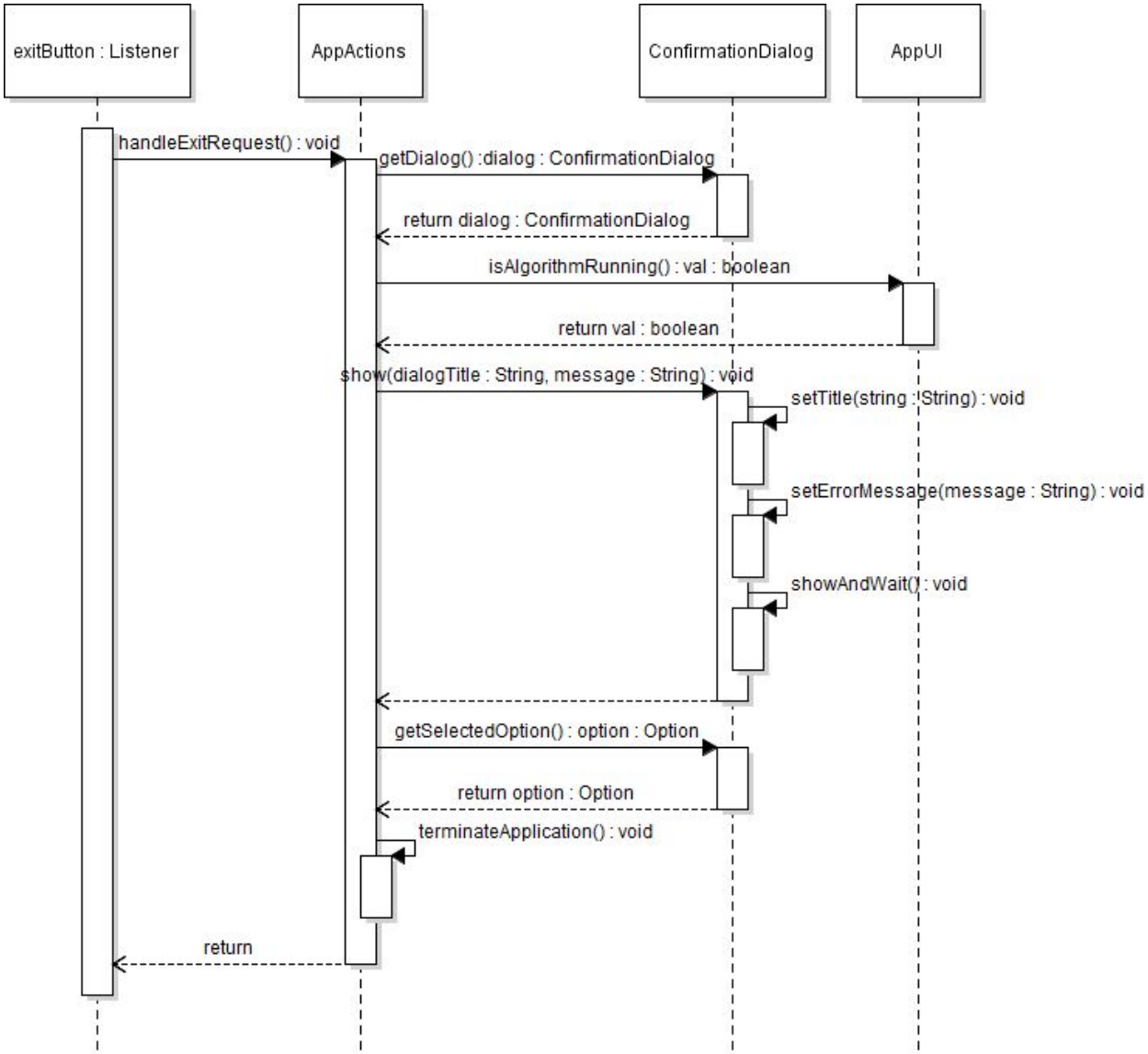



Diagram 5.14 : Exit Application(Use Case 10)

Now that we have handled the scenarios of having unsaved data, it is time to deal with exiting while an algorithm is running. Here there are only two valid options, terminating the algorithm and exiting the application or returning to the application. The sequence diagram in Diagram 5.14 is used for the scenario where the user wants to terminate the algorithm right away and close the application. Again, using the same method from the previous three diagram descriptions, we will prompt the user for a response after concluding that there is an algorithm running but we are assuming the user wants to terminate the application. Therefore once the response occurs, we simply call the terminateApplication method in AppActions which will end the application.

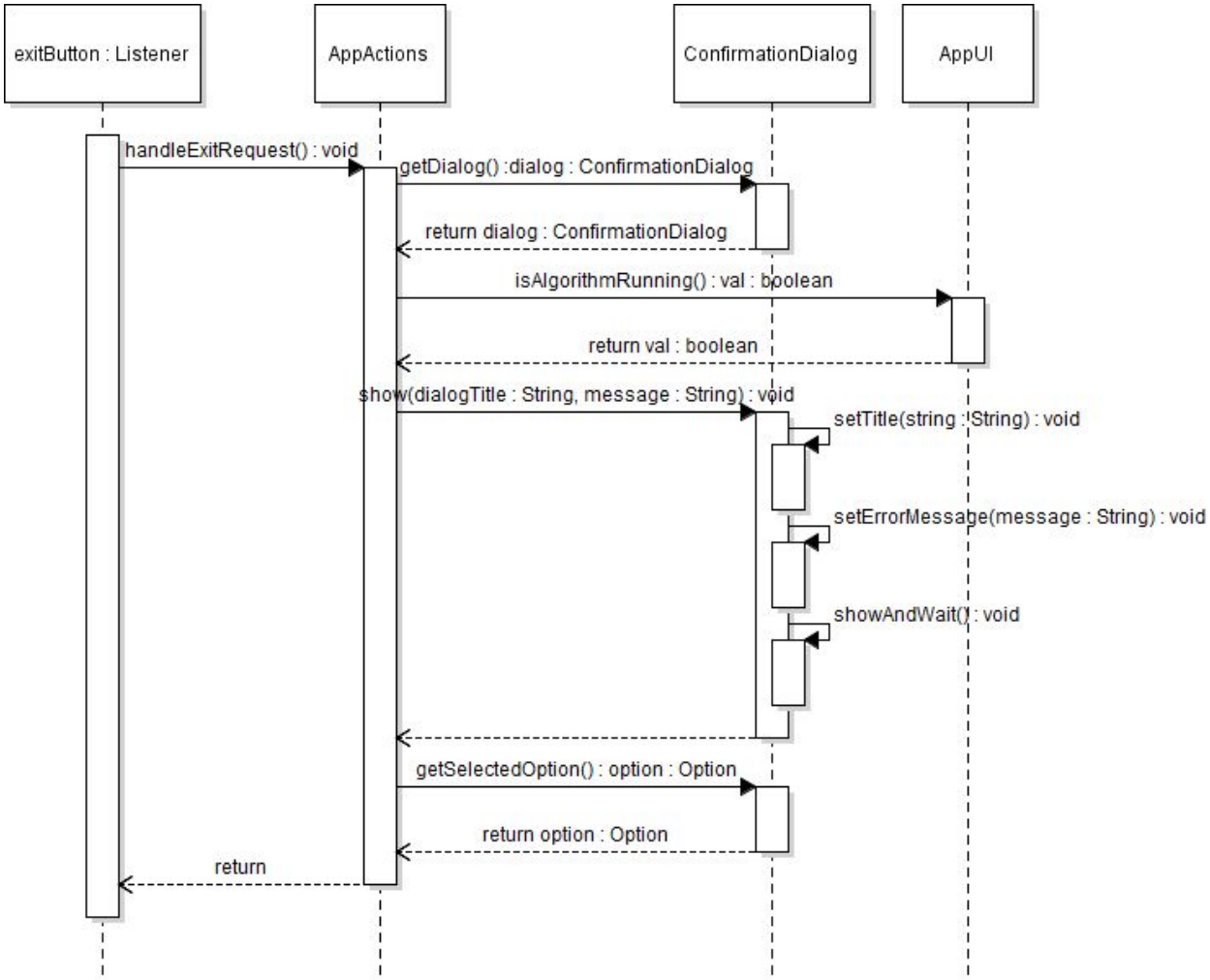



Diagram 5.15 : Exit Application(Use Case 10)

Finally, this scenario deals with having an algorithm running and clicking the exit button but wanting to return back to the application. Here, once the exitButton is clicked, the listener calls the handleExitRequest method in AppActions which checks if there is an algorithm running, then prompts the user to see what they would like to do. Once the user clicks that they would like to return, the methods simply return and the application continues iterating for the algorithm.

# 6. File/Data structures and formats

This section will include the File Structure that the application will be contained inside of alongside the formats that the properties.xml file should follow. All the created resources, including screenshots and data that is saved will be stored inside of data-vilij/resources/data.

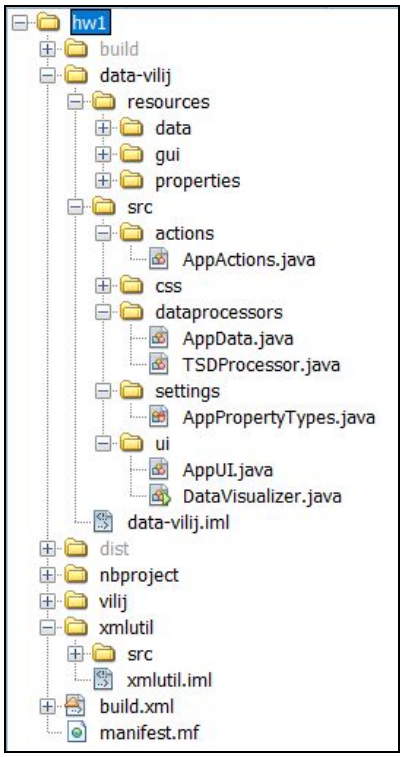



Image 5.1 : Application file structure

This application file structure must be maintained through development and distribution in order to provide consistency and reliability. Within this file structure, we see the main classes and important directories. The screenshots will be defaultly directed into the data directory inside data-vilij/resources alongside saved data entries. The XML file containing the name and value properties of data inside the application is stored under the properties directory in data-vilij/resources as well.

```xml
<properties>
    <property_list>
        <!-- RESOURCE FILES AND FOLDERS -->
        <property name="DATA_RESOURCE_PATH" value="data-vilij/resources/data"/>
        <property name="CSS_PATH" value="../css/LineGraph.css"/>
        <property name="SCREENSHOT_DIRECTORY" value="data-vilij/resources/data/Screenshot.png"/>

        <!-- USER INTERFACE ICON FILES -->
        <property name="SCREENSHOT_ICON" value="screenshot.png"/>

        <!-- TOOLTIPS FOR BUTTONS -->
        <property name="SCREENSHOT_TOOLTIP" value="Screenshot"/> <!-- will print current view of image to a file -->

        <!-- WARNING MESSAGES -->
        <property name="EXIT_WHILE_RUNNING_WARNING"
                  value="An algorithm is running. If you exit now, all unsaved changes will be lost. Are you sure?"/>
        <property name="EXIT_WHILE_RUNNING_WARNING_HEADING"
                  value="Confirm exit"/>
        <property name="TOO_MUCH_INFO_TITLE" value="Too much information"/>
        <property name="TOO_MUCH_INFO_ONE" value="The data loaded has "/>
        <property name="TOO_MUCH_INFO_TWO" value=" lines. Only the first 10 lines are being displayed in the text area."/>


        <!-- ERROR MESSAGES -->
        <property name="REFORMAT_ERROR" value="The data you have entered is invald, please reformat it."/>
        <property name="REFORMAT_ERROR_TITLE" value="Reformat Error"/>
        <property name="FILE_NOT_FOUND_ERROR" value="File not found, please reselect a file.\n"/>
        <property name="FILE_NOT_FOUND_ERROR_TITLE" value="File Not Found Error"/>
        <property name="RESOURCE_SUBDIR_NOT_FOUND" value="Directory not found under resources."/>

        <!-- APPLICATION-SPECIFIC MESSAGE TITLES -->
        <property name="SAVE_UNSAVED_WORK_TITLE" value="Save Current Work"/>

        <!-- APPLICATION-SPECIFIC MESSAGES -->
        <property name="SAVE_UNSAVED_WORK" value="Would you like to save current work?"/>

        <!-- APPLICATION-SPECIFIC PARAMETERS -->
        <property name="DATA_FILE_EXT" value="*.tsd"/>
        <property name="DATA_FILE_EXT_DESC" value="Tab-Separated Data File (.*.tsd)"/>
        <property name="FILE_CHOOSER_HEADING" value="Save the file as:"/>
        <property name="TEXT_AREA" value="Data File"/>
        <property name="SPECIFIED_FILE" value=" specified file"/>
        <property name="DISPLAY_BUTTON" value="Display"/>
        <property name="CHECKBOX_TEXT" value="Enable Read-Only"/>
        <property name="AXIS_STYLE" value="number_axis"/>
        <property name="CHART_STYLE" value="chart_style"/>
        <property name="FILE_TYPE" value="png"/>

        <property name="FILE_TYPE" value="png"/>
        <property name="UNIQUE_LETTER" value="@"/>
        <property name="BACKSPACE_STRING" value="BACKSPACE"/>
        <property name="DELETE_STRING" value="DELETE"/>

    </property_list>
    <property_options_list/>
</properties>
```

Image 5.1 : properties.xml format

Contains an accurate representation of how the properties.xml file should be formatted during development. Similarly, all XML files should hold this format to keep simplicity and functionality. Each property may or may not be used multiple times throughout the application but they are guaranteed to be used at least once.

# 7. Supporting Information

This document should serve as a reference for the designers and developers in the future stages of the development process. Since this product involves the use of a specific data format, we provide the necessary information in this section in the form of an appendix.

## Appendix A: Tab-Separated Data (TSD) Format

The data provided as input to this software as well as any data saved by the user as a part of its usage must adhere to the tab-separated data format specified in this appendix. The specification are as follows:

1. A file in this format must have the ".tsd" extension.
2. Each line (including the last line) of such a file must end with '\n' as the newline character.
3. Each line must consist of exactly three components separated by '\t'. The individual components are
    a. Instance name, which must start with '@'
    b. Label name, which may be null.
    c. Spatial location in the x-y plane as a pair of comma-separated numeric values. The values must be no more specific than 2 decimal places, and there must not be any whitespace between them.
4. There must not be any empty line before the end of file.
5. There must not be any duplicate instance names. It is possible for two separate instances to have the same spatial location, however.