DATAVILIJ™

Software Design Description

Author: Ivan Tinov
CSE 219 HW3
March 2018
Version 1.0

Abstract: This document describes the software design for DataViLiJ, a program that is developed to allow a user to load or input a specific data set and display it on a visualization tool to observe various trends in the data set.

Based on IEEE Std 830™ -1998 (R2009) document format

1 Table of Contents

2 Introduction

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

2.1 Purpose

The purpose of this document is to specify how the DATAVILIJ application should look and operate. This document serves as an agreement among all parties and as a reference for how the data visualization application should ultimately be constructed. Upon reading of this document, one should clearly understand the visual aesthetics and functionalities of application's user interface as well as understand the way AI algorithms are incorporated.

2.2 Scope

The goal of this project is for user attempting to learn about AI to have a visual understanding of the inner workings of the fundamental algorithms. AI is a vast field, and this project is limited to the visualization of two types of algorithms that "learn" from data. These two types are called **clustering** and **classification**. The design and development of these algorithms is outside the scope of the project, and the assumption is that such algorithms will already be developed independently, and their output will comply with the data format specified in this document. The DATAVILIJ application serves simply as a visualization tool for how those algorithms work. Both clustering and classification are, in theory, not limited to a fixed number of labels for the data, but this project will be limited to at most four labels for clustering algorithms, and exactly two labels for classification algorithms. Further, the design and development of this project will also assume that the data is 2-dimensional. This means that 3D visualization is currently beyond the scope of the DATAVILIJ application. Similarly, touch screen interaction with the GUI is also not possible.

2.3 Definitions, Acronyms, and Abbreviations

The section below contains important definitions that will help with the understanding of the way the program will work, what it will incorporate, and how it will be made.

1) **Algorithm:** In this document, the term 'algorithm' will be used to denote an AI algorithm that can "learn" from some data and assign each data point a label.

2) **Clustering:** A type of AI algorithm that learns to assign labels to instances based purely on the spatial distribution of the data points.

3) **Classification:** A type of AI algorithm that learns to assign new labels to instances based on how older instances were labeled. These algorithms calculate geometric objects that divide the *x*-*y* plane into parts. E.g., if the geometric object is a circle, the two parts are the *inside* and the *outside* of that circle; if the geometric object is a straight-line, then again, there two parts, one on each side of the line.

4) **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.

5) **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.

6) **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.

7) **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.

8) **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.

9) **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.

10) **Unified Modeling Language (UML):** A general-purpose, developmental modeling language to provide a standard way to visualize the design of a system.

11) **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.

12) **User:** Someone who interacts with the DataViLiJ application via its GUI.

13) **User Interface (UI):** See *Graphical User Interface (GUI)*.

2.4 References

(1) IEEE Software Engineering Standards Committee. "IEEE Recommended Practice for Software Requirements Specifications." In IEEE Std. 830-1998, pp. 1-40, 1998.

(2) IEEE Software Engineesring Standards Committee. "IEEE Standard for Information Technology – Systems Design – Software Design Descriptions." In IEEE STD 1016-2009, pp.1-35, July 20, 2009

(3) Rumbaugh, James, Ivar Jacobson, and Grady Booch. Unified modeling language reference manual, The. Pearson Higher Education, 2004.

(4) McLaughlin, Brett, Gary Pollice, and David West. Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D. O'Reilly Media, Inc., 2006.

(5) Riehle, Dirk, and Thomas Gross. "Role model based framework design and integration." In ACM SIGPLAN
Notices, Vol. 33, No. 10, pp. 117-133. ACM, 1998.

(6) DATAVILIJ™ SRS – Professaur Inc's Software Requirements Specification for the DATAVILIJ application.

2.5 Overview

This software design description (SDD) document will define the path that this project will take and provides a working design for the DATAVILIJ software application as described in the DATAVILIJ software requirements specification (SRS). It is important to note that all parties involved in the implementation stage must agree upon all connections between components before proceeding to the implementation stage. As an overview of the entire document, Section 1 above provides a Table of Contents. Section 3 will provide a Package-Level Viewpoint that includes an overview of the software used, as well as a description of Java packages and frameworks to be used. Section 4 will provide a Class-Level Design Viewpoint, showcasing the

way various classes will be constructed using UML Class Diagrams. Section 5 will provide the Method-Level Design Viewpoint that will describe the task of each method as well as how the methods in the program interact with each other. Section 6 will provide the File/Data Structure and formats that are used in the program to make the various specifications required work.

3 Package-Level Design Viewpoint

This section will encompass the design of the DATAVILIJ application, by giving an overview of the software used such as Java packages and frameworks used and implemented. In constructing this application, there will be heavy reliance on the Java API, the VILIJ framework that the DATAVILIJ is built based upon, as well as XMLUtil package that provides general utility and helper methods for interacting with various XML data. Below are descriptions of the components that will be constructed, as well as how everything described here will be used to build the DATAVILIJ application.

3.1 DATAVILIJ Software Overview

The DATAVILIJ application will be designed with major extensions to the basic VILIJ framework that is provided as a foundation to the project. Figure 3.1 below specifies all the components of the data-vilij source code implementations and the various packages that each class is part of.

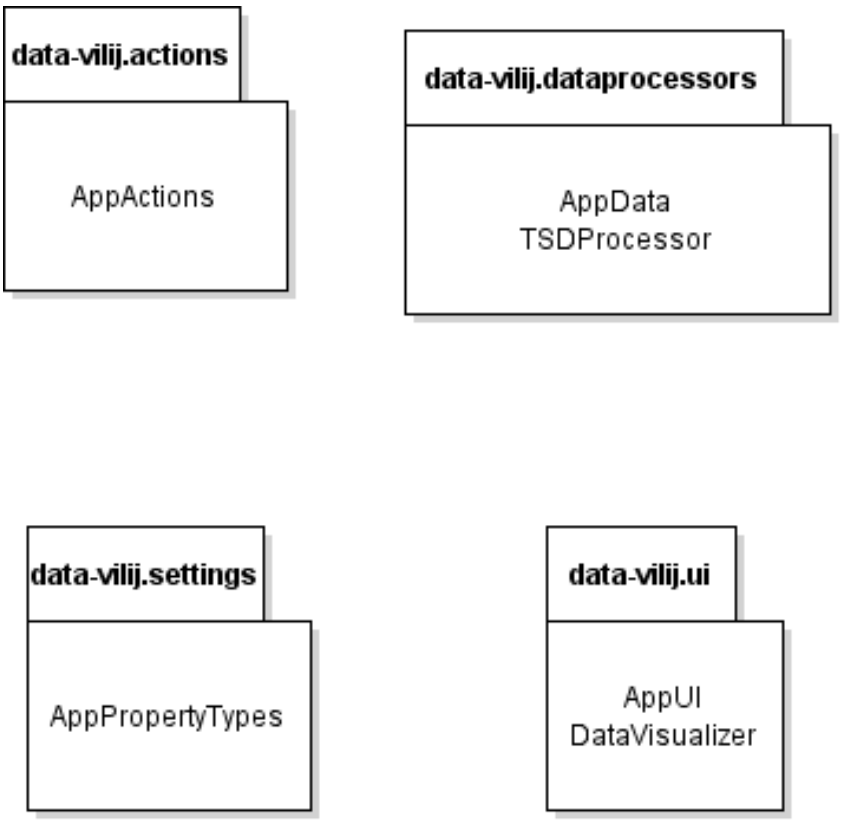

Figure 3.1.1: DATAVILIJ Package Overview

The above Figure gives a general description for the DATAVILIJ base framework package-level design.

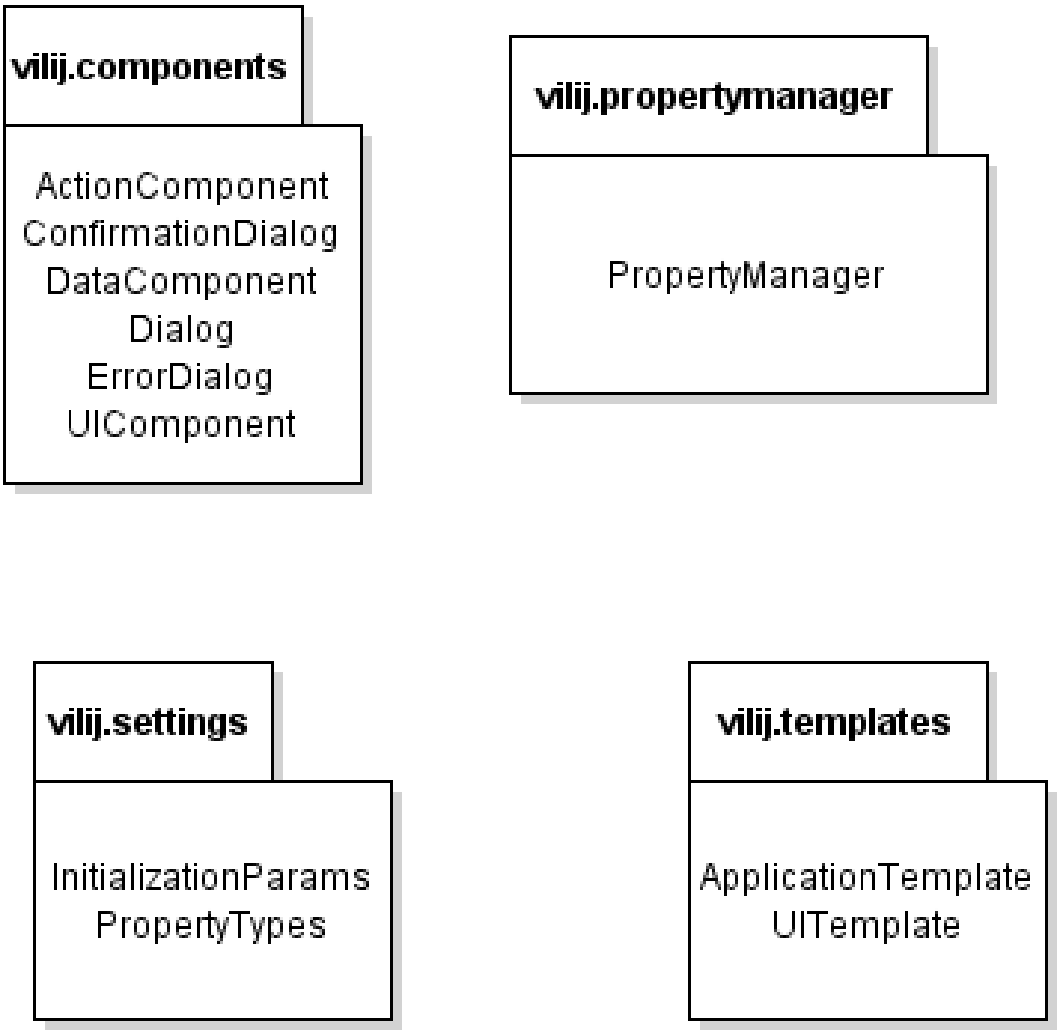

Figure 3.1.2: VILIJ Framework Package Overview

The above Figure gives a general description for the VILIJ base framework package-level design.

3.2 Java API Usage

The DATAVILIJ application will make use of the various classes showcased in Figure 3.2 and will be developed using the Java programming language.

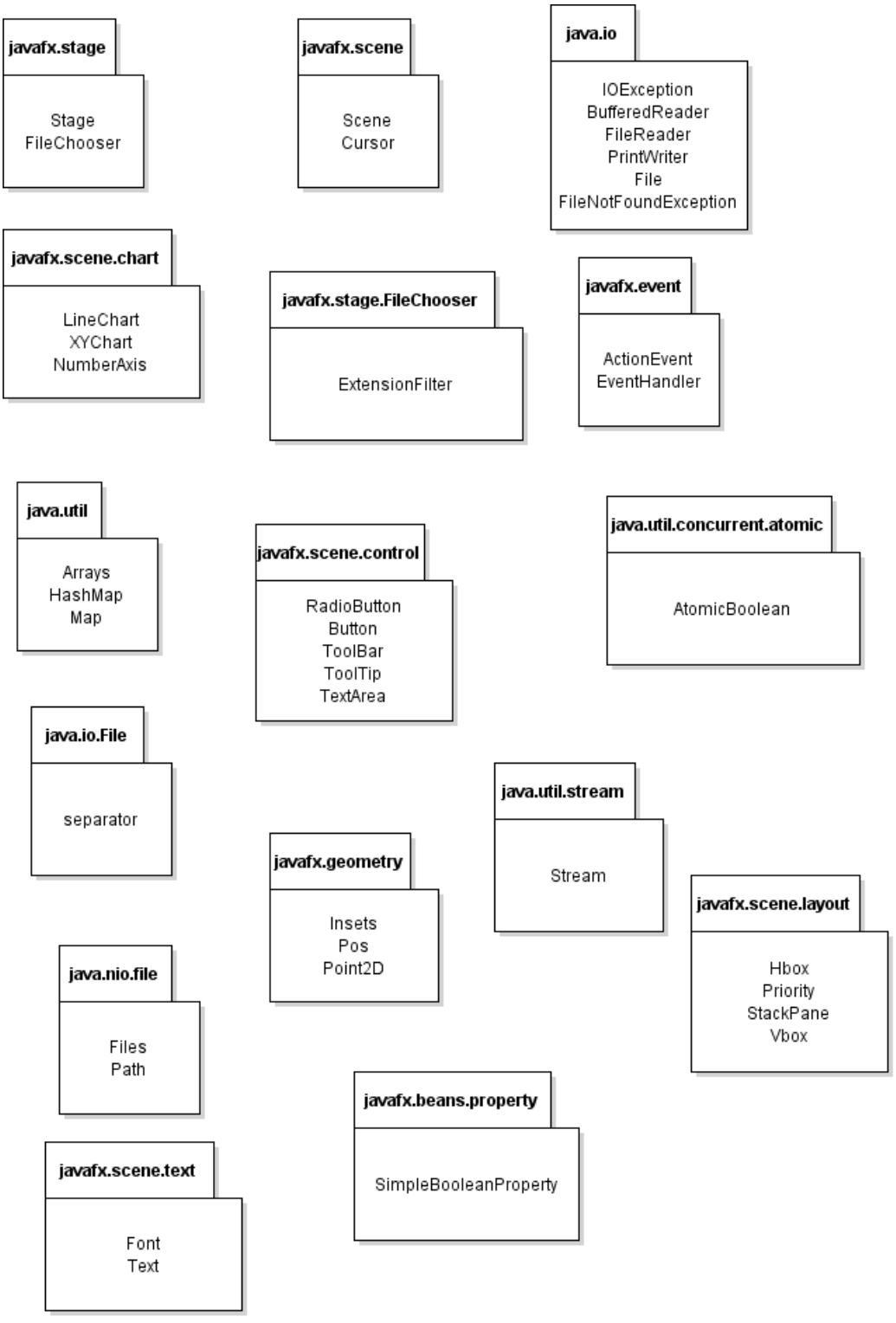

Figure 3.2: Java API Classes and Packages Used

The above figure represents all the Java API package and resources used to aid in creating the DATAVILIJ application

3.3 Java API Usage Descriptions

Tables 3.3.1 – 3.3.16 below summarize the way each class described above will be used.

javafx.stage	Uses
Stage	Sets the primaryStage for which a JavaFX application runs on.
FileChooser	Provides standard support for opening a file save, load, choose dialog.

Table 3.3.1: Uses the classes found in Java API's javafx.scene.control

javafx.scene	Uses
Scene	Sets the scene for a JavaFX application on which various nodes can be placed.
Cursor	Used for setting the action and behavior of the mouse.

Table 3.3.2: Uses the classes found in Java API's javafx.scene

java.io	Uses
IOException	Sends a signal that some sort of I/O exception has occurred.
BufferedReader	Used to read the text from a character-input stream, buffering the characters to provide an efficient way of reading characters, arrays, and lines.
FileReader	Used for reading character files.
PrintWriter	Used for printing formatted representations of objects to a text-output stream.
File	An abstract representation of file and directory pathnames.
FileNotFoundException	Exception that signals that an attempt to open a file represented by a specific pathname has failed.

Table 3.3.3: Uses the classes found in Java API's java.io

javafx.scene.chart	Uses
LineChart	Used to create a chart that plots a line according to data points in a specified series of coordinates.
XYChart	Chart base class for 2 axis charts. Responsible for drawing the two axis and the plot content.
NumberAxis	An axis class that plots a range of numbers with major tick marks along the x-axis and the y-axis of a chart.

Table 3.3.4: Uses the classes found in Java API's javafx.scene.chart

javafx.stage.FileChooser	Uses
ExtensionFilter	Defines an extension filter that is used for filtering which files can be chosen in a FileDialog based on the file name extensions.

Table 3.3.5: Uses the classes found in Java API's javafx.stage.FileChooser

javafx.event	Uses
ActionEvent	An event that is used to represent some kind of action such as when a node in the scene has been interacted with. Ex. A button being clicked.
EventHandler	A functional interface that handles events of a specified class/type.

Table 3.3.6: Uses the classes found in Java API's javafx.event

java.util	Uses
Arrays	Contains various methods for manipulating arrays, such as sorting and searching.
Map	An object that maps keys to values.
HashMap	Provides basic implementation of Map interface and stored data in (key, value) pairs. Used in processing the text from the text area into a format that the chart can understand.

Table 3.3.7: Uses the classes found in Java API's java.util

javafx.scene.control	Uses
RadioButton	Used as a checkbox for making the textarea read-only.
Button	Creates a GUI button to be clicked and produce a response via the EventHandler.
ToolBar	A control which displays items horizontally or vertically. In our case, a toolbar is used to hold the new, save, open, print, exit, screenshot buttons on the top of the application GUI.
ToolTip	A UI element that is used to display the instance name, label, and coordinates of a point on the chart.
TextArea	Text input component that allows a user to enter multiples lines of plain text.

Table 3.3.8: Uses the classes found in Java API's javafx.scene.control

java.util.concurrent.atomic	Uses
AtomicBoolean	Used as a boolean value that may be read and written atomically.

Table 3.3.9: Uses the classes found in Java API's java.util.concurrent.atomic

java.io.File	Uses
Separator	Used to separate a pathname by a default separator character.

Table 3.3.10: Uses the classes found in Java API's java.io.File

javafx.geometry	Uses
Insets	Used in setting the padding of a node in the GUI, such as the text area, and the chart.
Pos	A set of values that describe the vertical and horizontal positioning and alignment of a node in the scene.
Point2D	A 2D geometric point that usually represents the x,y coordinates of a graph.

Table 3.3.11: Uses the classes found in Java API's javafx.geometry

java.util.Stream	Uses
Stream	Used as a way of processing the data placed into the text area or loaded from a saved text area. A stream represents a sequence of objects from a source.

Table 3.3.12: Uses the classes found in Java API's java.util.Stream

javafx.scene.layout	Uses
Priority	Enumeration that is used to determine the growth or shrink in priority of a given node's layout area.
Hbox	A node that extends Pane and lays out its children in a horizontal row.
Vbox	A node that extends Pane and lays out its children in a vertical column.
StackPane	A node that extends Pane and lays its children in a back-to-front stack.

Table 3.3.13: Uses the classes found in Java API's javafx.scene.layout

java.nio.file	Uses
Files	A class that consist of static methods that operate on files, directories or other types of files.
Path	An object that is used to locate a file in a file system. Used to specify the path directory to open or save a file.

Table 3.3.14: Uses the classes found in Java API's java.nio.file

javafx.scene.text	Uses
Font	Used to change/edit the font of a selected text
Text	Used for editing the labels that represent the text area, chart, and read-only radio button.

Table 3.3.15: Uses the classes found in Java API's javafx.scene.text

javafx.beans.property	Uses
SimpleBooleanProperty	Used to bind nodes together with a Boolean value.

Table 3.3.16: Uses the classes found in Java API's javafx.beans.property

4 Class-Level Design Viewpoint

This section will encompass the class design of the DATAVILIJ application.

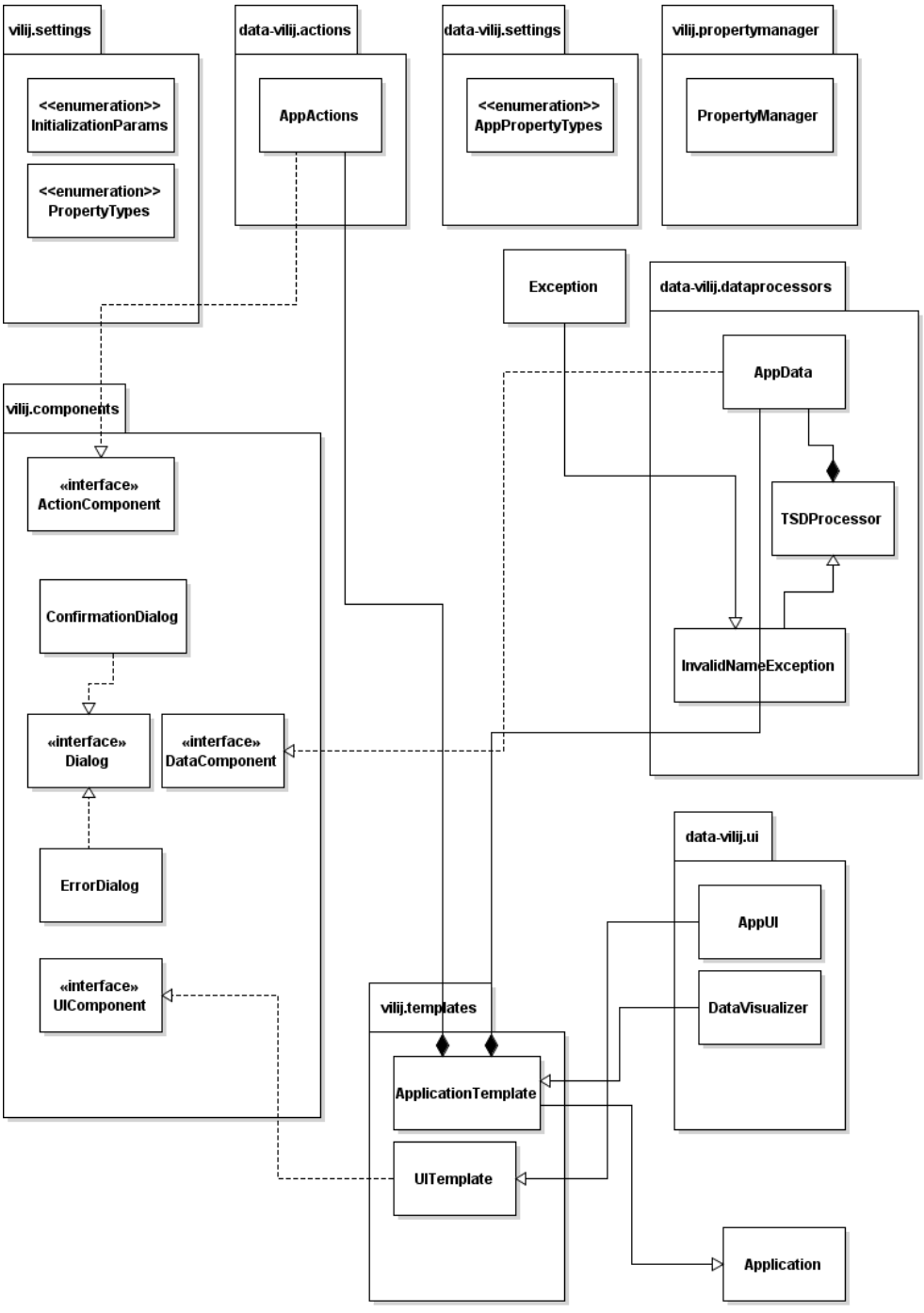

Figure 4.1: DATAVILIJ application entire class UML Diagram

In the figure above, the relationship and interaction between the various classes is represented. For more information about the variables, and methods represented in each class part of the DATAVILIJ application, look at figures 4.2-4.6 below

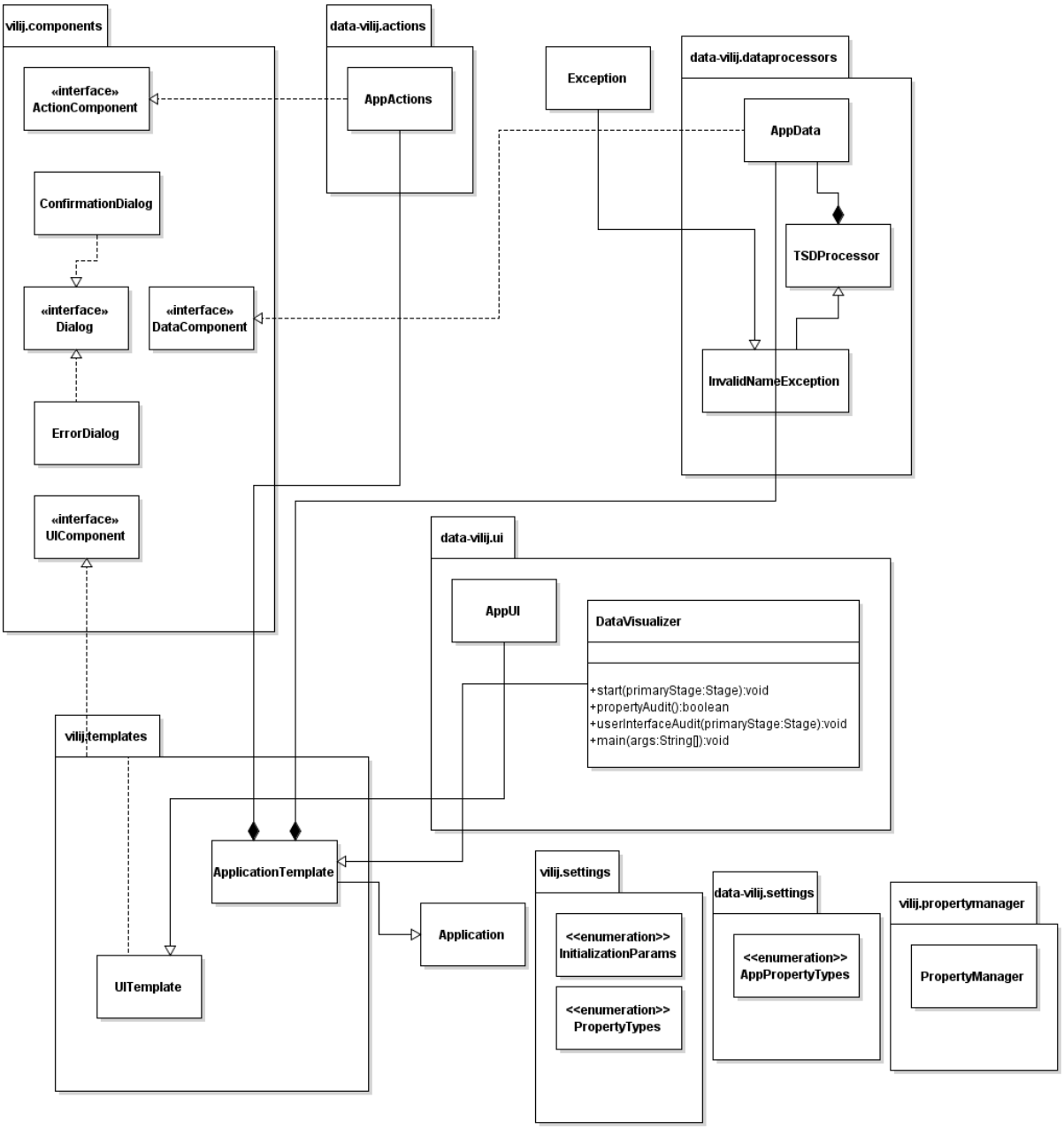

Figure 4.2: DATAVILIJ application DataVisualizer class UML Diagram

The DataVisualizer class is what launches our application and has the necessary start and audit methods to initialize the GUI and the base framework.

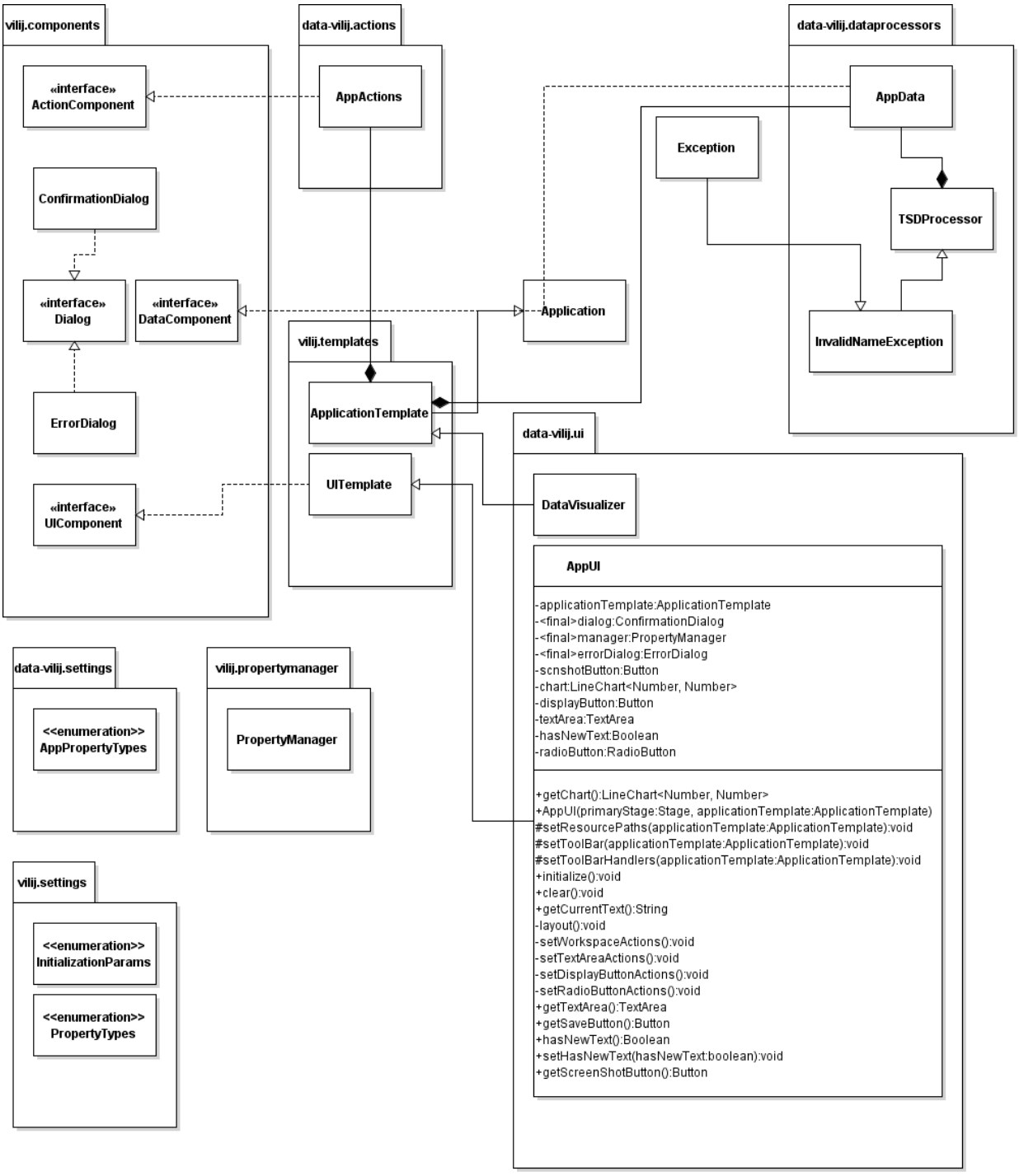

Figure 4.3: DATAVILIJ application AppUI class UML Diagram

The AppUI class is what holds all the variables and methods for setting up the custom UI, as well as listeners for interaction with the UI. There is also communication between the UI created solely for the DATAVILIJ application with the base UI that is implemented in the framework.

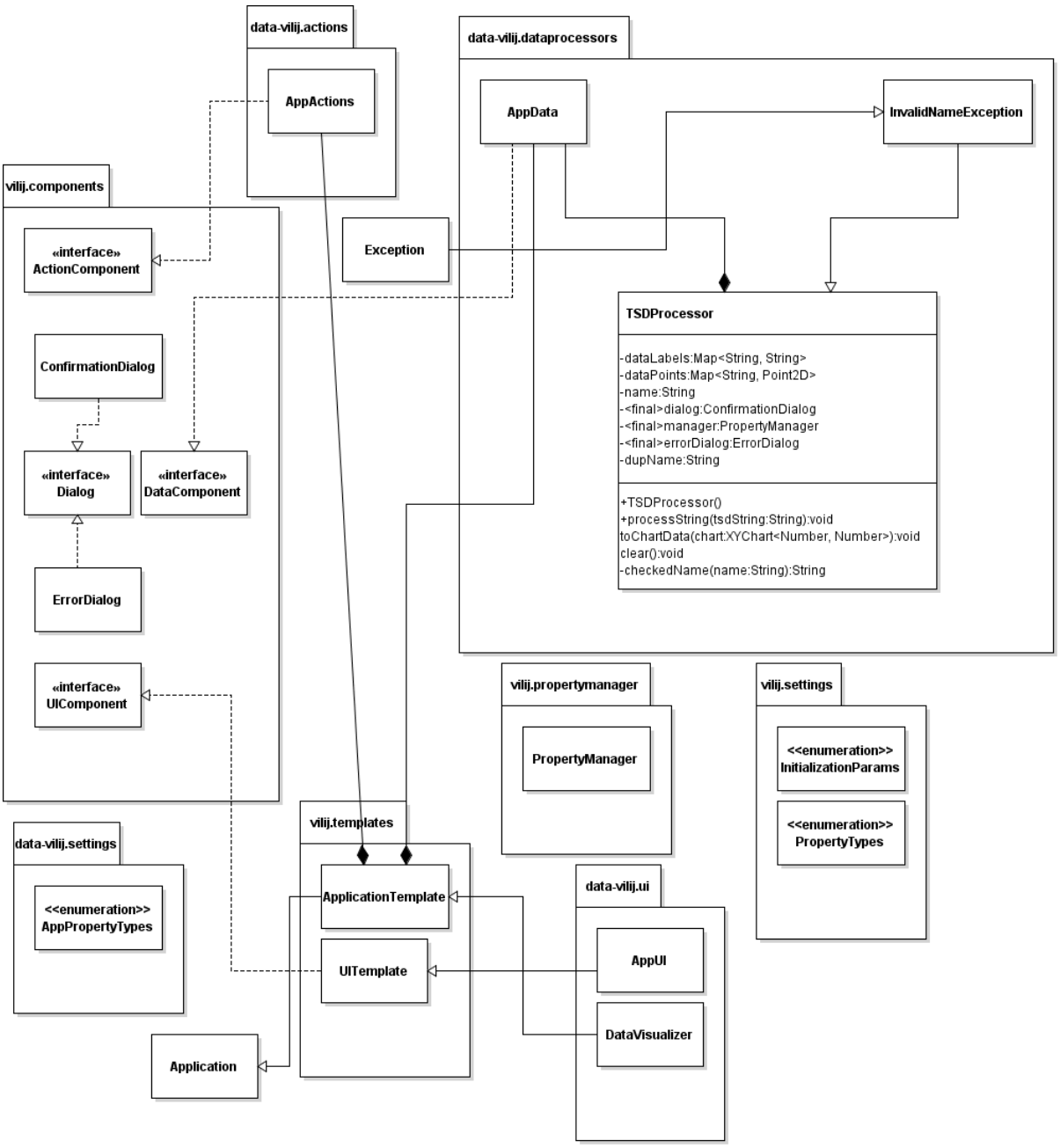

Figure 4.4: DATAVILIJ application TSDProcessor class UML Diagram

The TSDProcessor class contains very important methods for taking in the input from the TextArea and breaking down the text to see if it matched the specific requirements for text input. If the input is correct, there is a method for taking the processed data and displaying it on the chart.

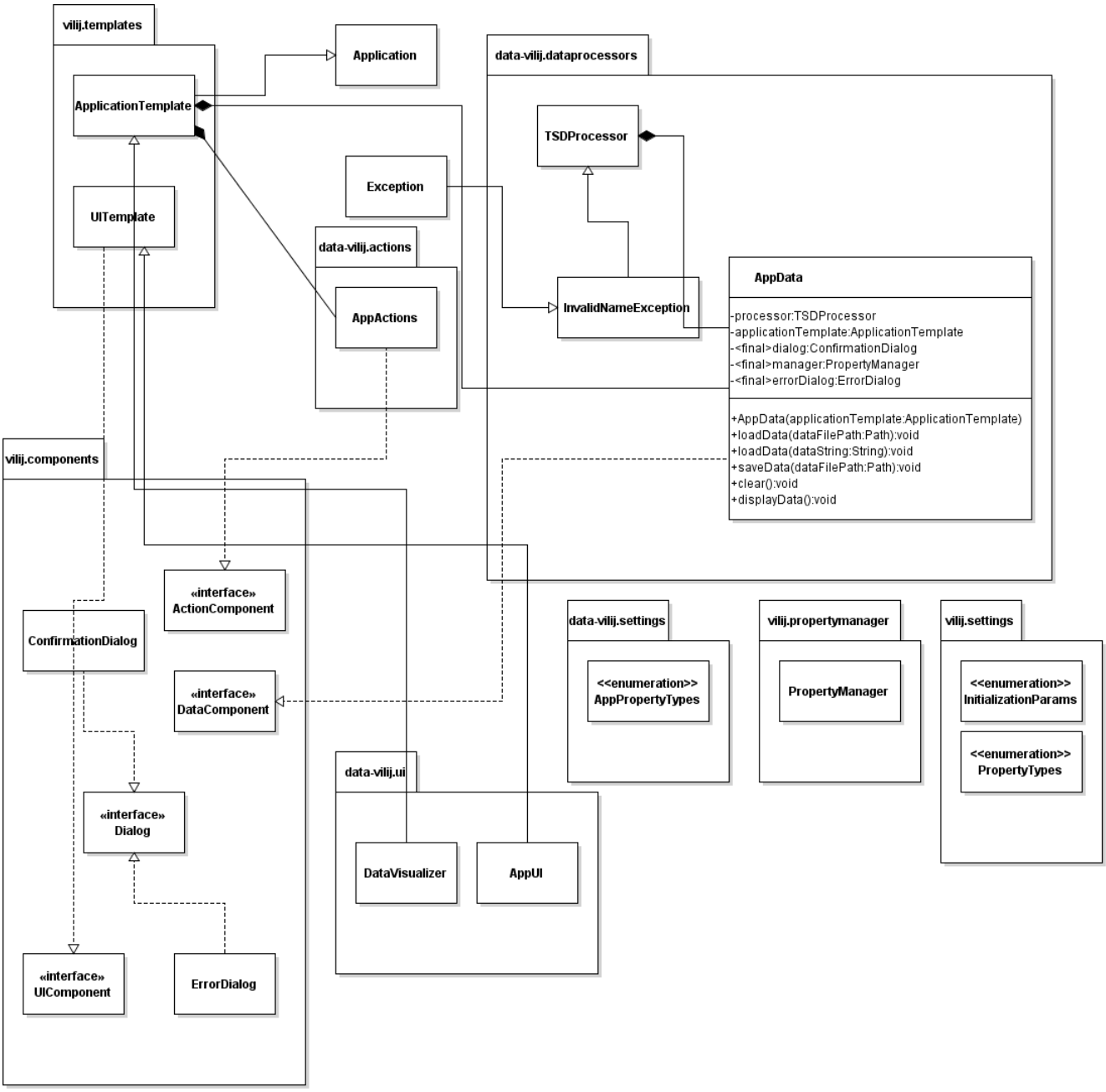

Figure 4.5: DATAVILIJ application AppData class UML Diagram

The AppData class contains the necessary methods for communication with the AppUI and the TSDProcessor class for loading data, saving data, displaying data, and clearing data. There are two types of load data methods, one that loads data from the TextArea as a string and another that loads the data from a path that points to somewhere on the user's computer.

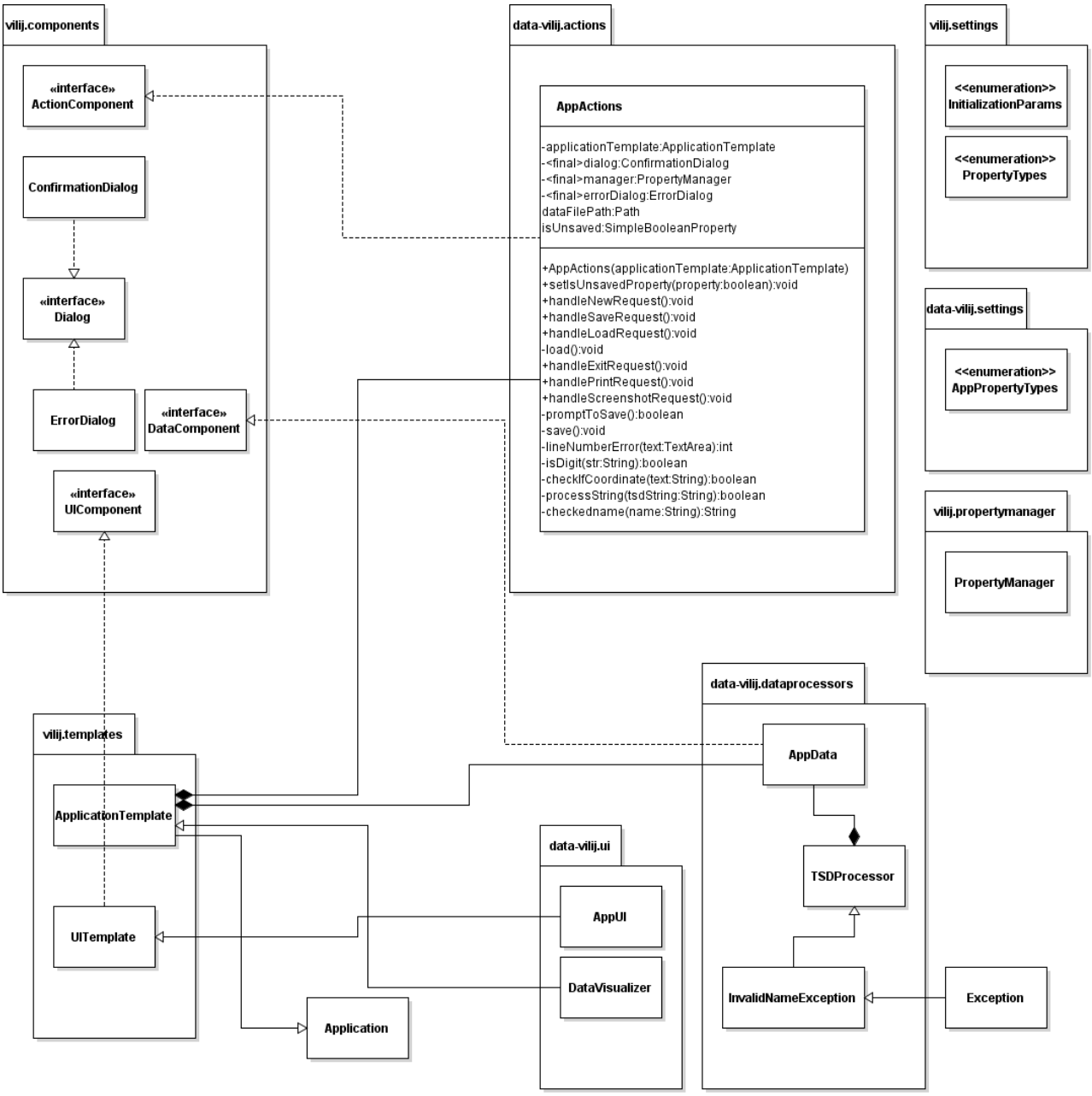

Figure 4.6: DATAVILIJ application AppActions class UML Diagram

The AppActions method is a core method where the AppData, and AppUI act together to handle methods associated with the loading, saving, and displaying of data. The load and save handle methods have specific implementations that check whether the data format is valid before loading data, saving data, and displaying data. Use of error handling is evident to control these errors and produce a correct error message when there is an error. Various other helper methods are included to check the data for validity.

5 Method-Level Design Viewpoint

This section will encompass the method design of the DATAVILIJ application.

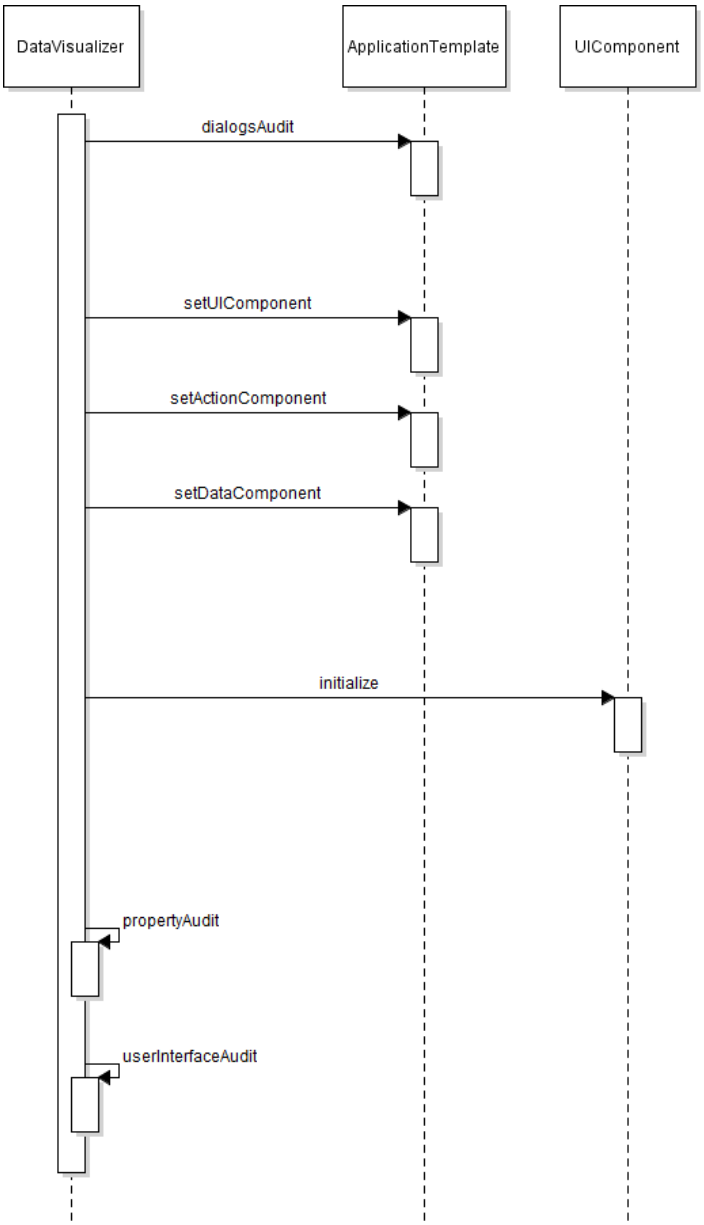

Figure 5.1: DATAVILIJ application UML Sequence Diagram for DataVisualizer

The above figure describes the method calls for the entire DataVisualizer class to show how the methods interact with the class itself as well as other classes that are associated with the DataVisualizer class.

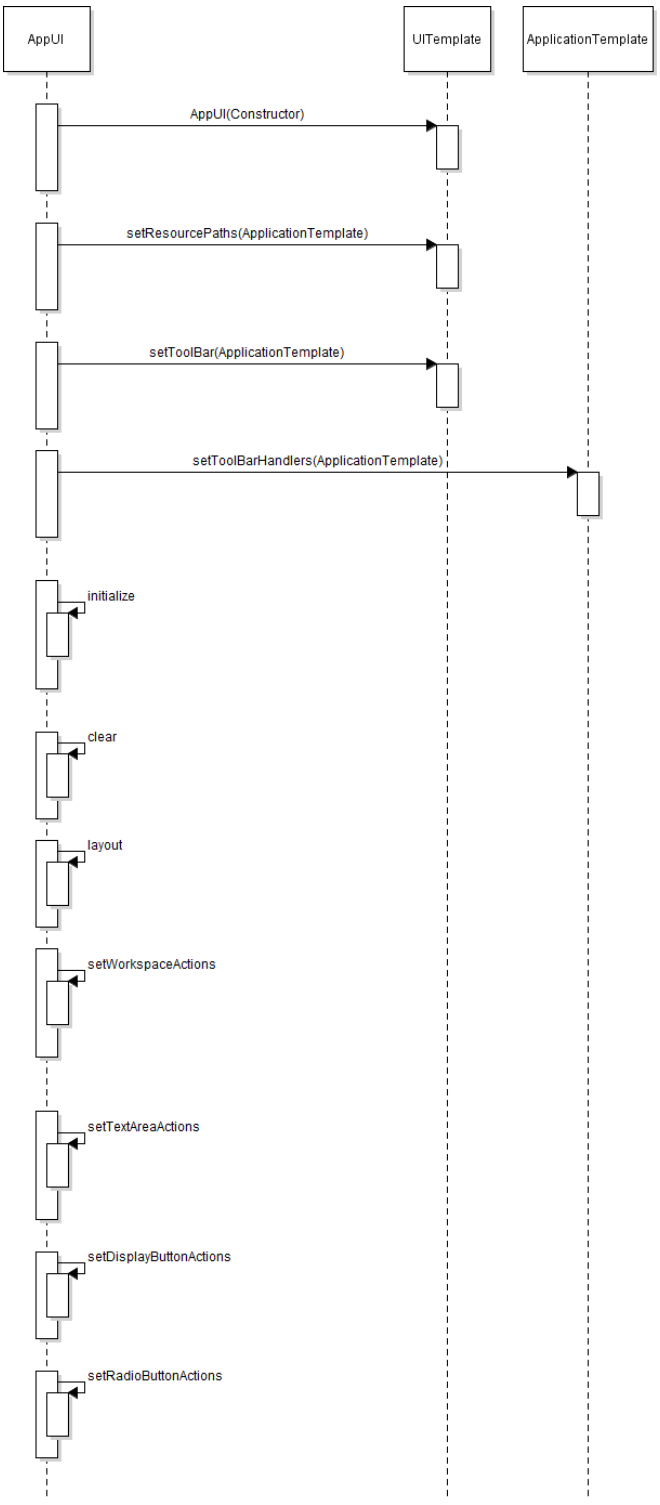

Figure 5.2: DATAVILIJ application UML Sequence Diagram for AppUI

The above figure describes the method calls for the entire AppUI class to show how the methods interact with the class itself as well as other classes that are associated with the AppUI class.

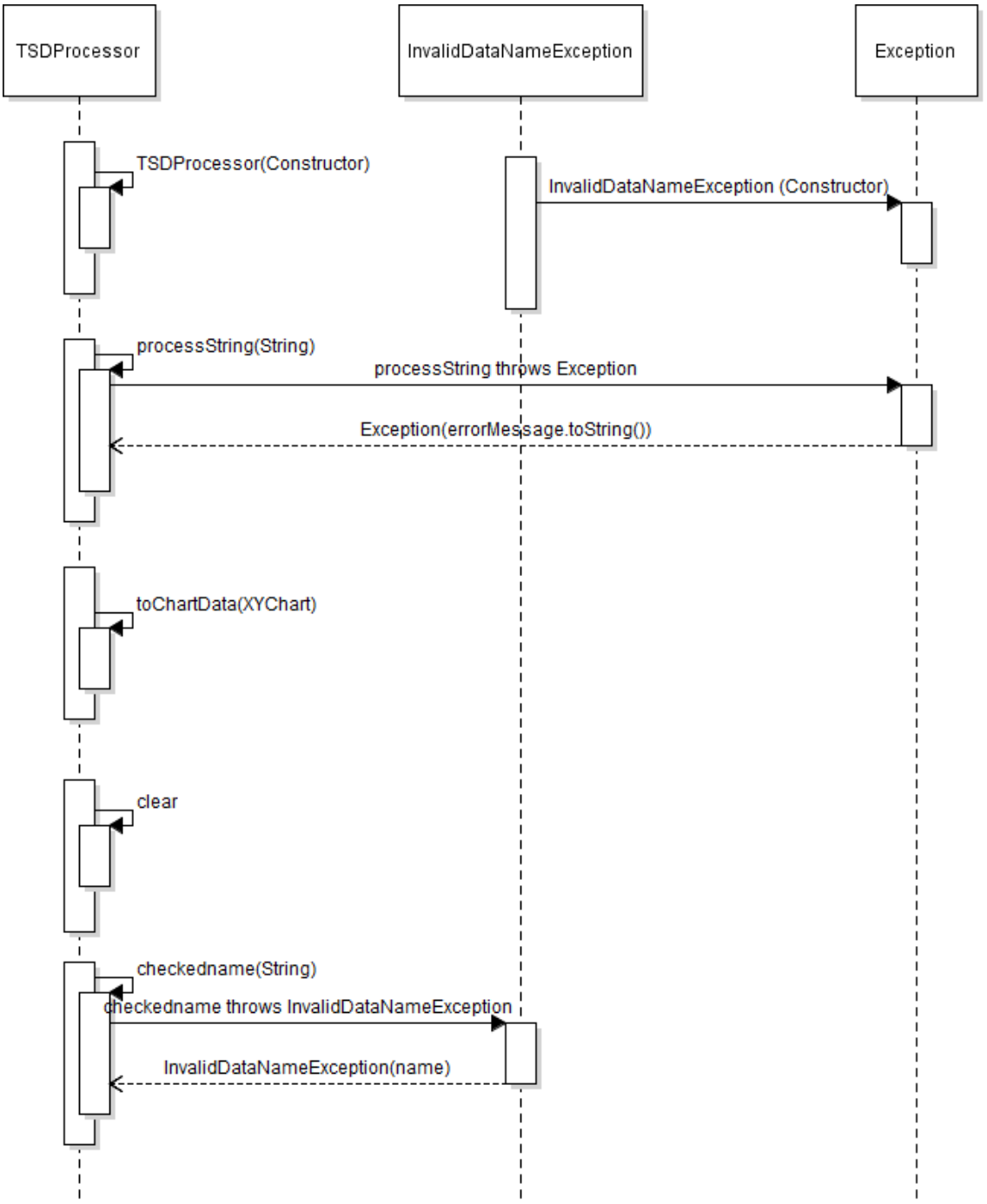

Figure 5.3: DATAVILIJ application UML Sequence Diagram for TSDProcessor

The above figure describes the method calls for the entire TSDProcessor class to show how the methods interact with the class itself as well as other classes that are associated with the TSDProcessor class.

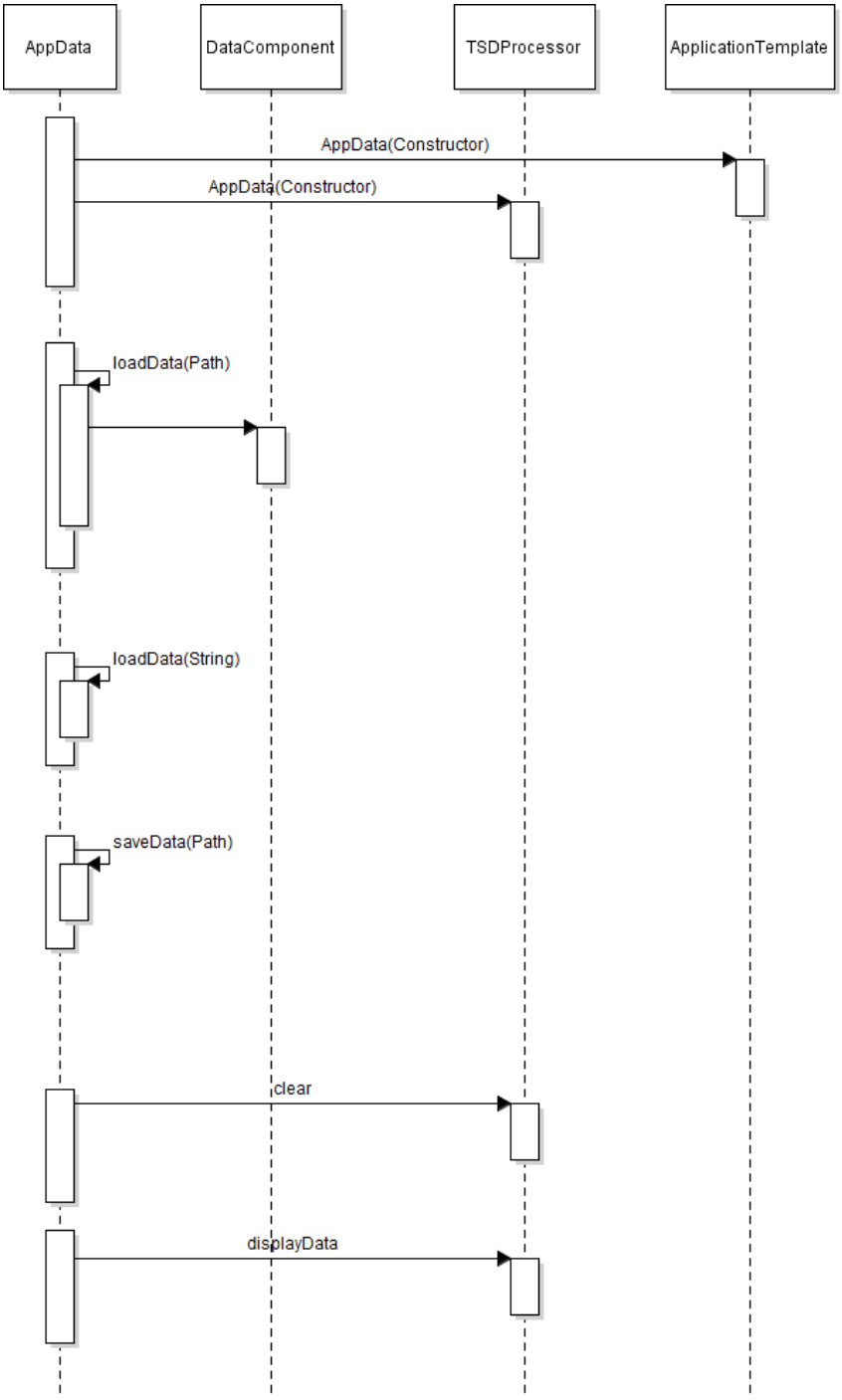

Figure 5.4: DATAVILIJ application UML Sequence Diagram for AppData

The above figure describes the method calls for the entire AppData class to show how the methods interact with the class itself as well as other classes that are associated with the AppData class.

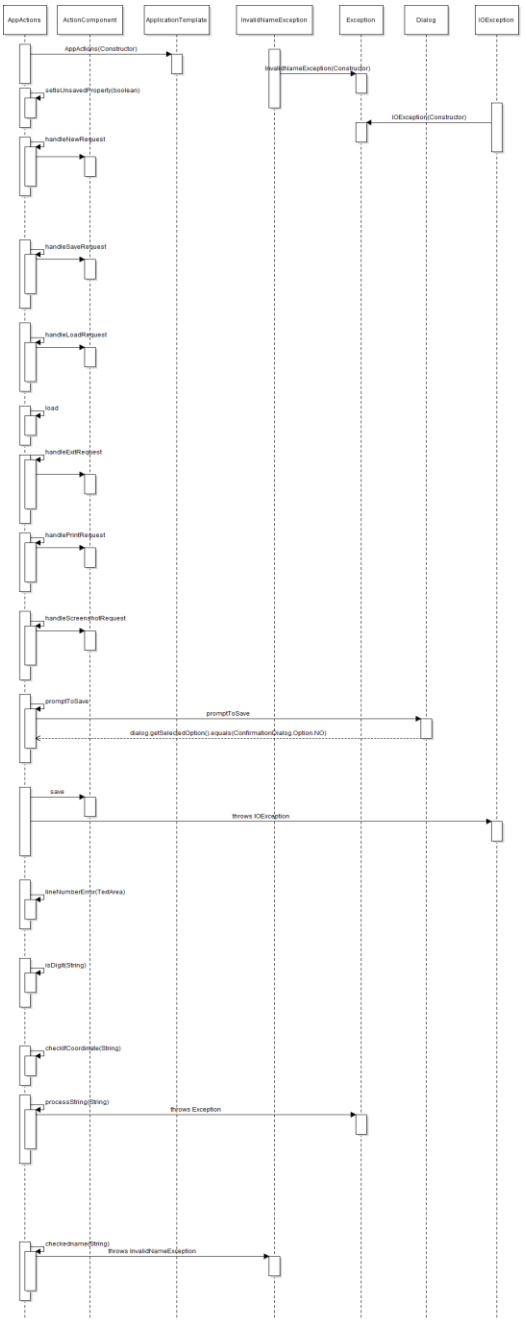

Figure 5.5: DATAVILIJ application UML Sequence Diagram for AppActions

The above figure describes the method calls for the entire AppActions class to show how the methods interact with the class itself as well as other classes that are associated with the AppActions class.

6 File/Data Structures and Formats

The file structure of DATAVILIJ application is represented below with the base framework classes under vilij\src. The core implementation of the DATAVILIJ application is under data-vilij\src.

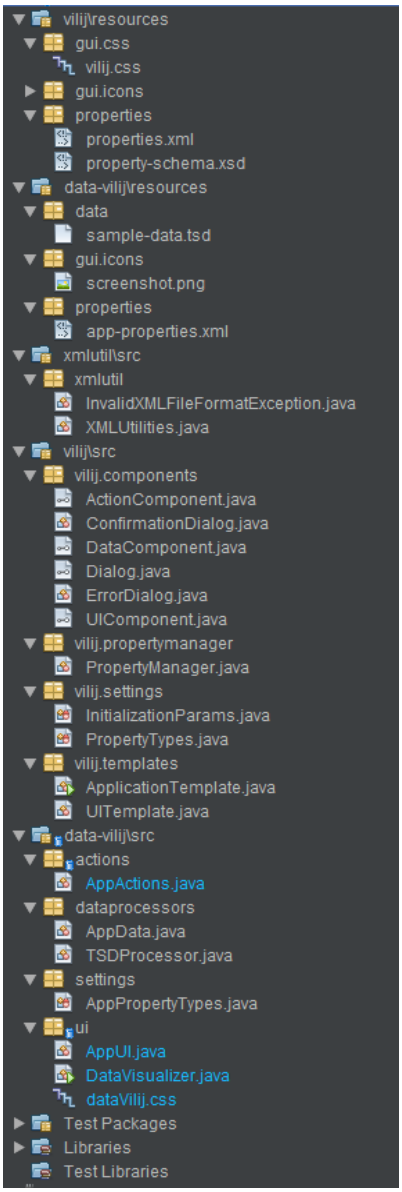

Figure 6.1: DATAVILIJ program and file structure

```
<properties>
    <property_list>
        <!-- RESOURCE FILES AND FOLDERS -->
        <property name="DATA_RESOURCE_PATH" value="data"/>

        <!-- 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?"/>

        <!-- ERROR MESSAGES -->
        <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"/>
        <property name="TEXT_AREA" value="text area"/>
        <property name="SPECIFIED_FILE" value=" specified file"/>
        <property name="LEFT_PANE_TITLE" value="Data File"/>
        <property name="LEFT_PANE_TITLEFONT" value="SansSerif"/>
        <property name="LEFT_PANE_TITLESIZE" value="18"/>
        <property name="CHART_TITLE" value="Data Visualization"/>
        <property name="DISPLAY_BUTTON_TEXT" value="Display Data"/>

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

Figure 6.2: app-properties.xml format

This represents how the application properties should look and contains various information application specific resource files, icons, tooltips, warning and error messages, titles, application messages, and parameters.

```
<properties>
    <property_list>
        <!-- HIGH-LEVEL USER INTERFACE PROPERTIES -->
        <property name="WINDOW_WIDTH" value="1000"/>
        <property name="WINDOW_HEIGHT" value="750"/>
        <property name="IS_WINDOW_RESIZABLE" value="false"/>
        <property name="TITLE" value="Visualization Library In Java (Vilij)"/>

        <!-- RESOURCE FILES AND FOLDERS -->
        <property name="GUI_RESOURCE_PATH" value="gui"/>
        <property name="CSS_RESOURCE_PATH" value="css"/>
        <property name="CSS_RESOURCE_FILENAME" value="vilij.css"/>
        <property name="ICONS_RESOURCE_PATH" value="icons"/>

        <!-- USER INTERFACE ICON FILES -->
        <property name="NEW_ICON" value="new.png"/>
        <property name="PRINT_ICON" value="print.png"/>
        <property name="SAVE_ICON" value="save.png"/>
        <property name="SAVED_ICON" value="saved.png"/>
        <property name="LOAD_ICON" value="load.png"/>
        <property name="EXIT_ICON" value="exit.png"/>
        <property name="LOGO" value="logo.png"/>

        <!-- TOOLTIPS FOR BUTTONS -->
        <property name="NEW_TOOLTIP" value="Create new data"/>
        <property name="LOAD_TOOLTIP" value="Load data from file"/>
        <property name="PRINT_TOOLTIP"
                  value="Print visualization"/> <!-- will save original image to a file, irrespective of zoom in/out view -->
        <property name="SAVE_TOOLTIP" value="Save current data"/>
        <property name="EXIT_TOOLTIP" value="Exit application"/>

        <!-- ERROR TITLES -->
        <property name="NOT_SUPPORTED_FOR_TEMPLATE_ERROR_TITLE" value="Operation Not Supported for Templates"/>
        <property name="LOAD_ERROR_TITLE" value="Load Error"/>
        <property name="SAVE_ERROR_TITLE" value="Save Error"/>

        <!-- ERROR MESSAGES FOR ERRORS THAT REQUIRE AN ARGUMENT -->
        <property name="PRINT_ERROR_MSG" value="Unable to print to "/>
        <property name="SAVE_ERROR_MSG" value="Unable to save to "/>
        <property name="LOAD_ERROR_MSG" value="Unable to load from "/>

        <!-- STANDARD LABELS AND TITLES -->
        <property name="CLOSE_LABEL" value="Close"/>
        <property name="SAVE_WORK_TITLE" value="Save"/>
    </property_list>
    <property_options_list/>
</properties>
```

Figure 6.3: properties.xml format

This contains the properties of the entire application such as resource files, higher level interface properties, icon files, tooltips for buttons, error titles, error messages, and standard labels for the base DATAVILIJ application. This is a broader properties format for the base framework.

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="properties">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="property_list">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="property" maxOccurs="unbounded" minOccurs="0">
                                <xs:complexType>
                                    <xs:simpleContent>
                                        <xs:extension base="xs:string">
                                            <xs:attribute type="xs:string" name="name" use="required"/>
                                            <xs:attribute type="xs:string" name="value" use="required"/>
                                        </xs:extension>
                                    </xs:simpleContent>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="property_options_list" maxOccurs="1" minOccurs="0">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="property_options" maxOccurs="unbounded" minOccurs="0">
                                <xs:complexType>
                                    <xs:sequence>
                                        <xs:element type="xs:string" name="option" maxOccurs="unbounded" minOccurs="0"/>
                                    </xs:sequence>
                                    <xs:attribute type="xs:string" name="name"/>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Figure 6.4: property-schema.xml format

This represents the model or outline for the properties of the DATAVILIJ application.

Figure 6.5: DATAVILIJ application icons used for the various buttons and interactions in the UI

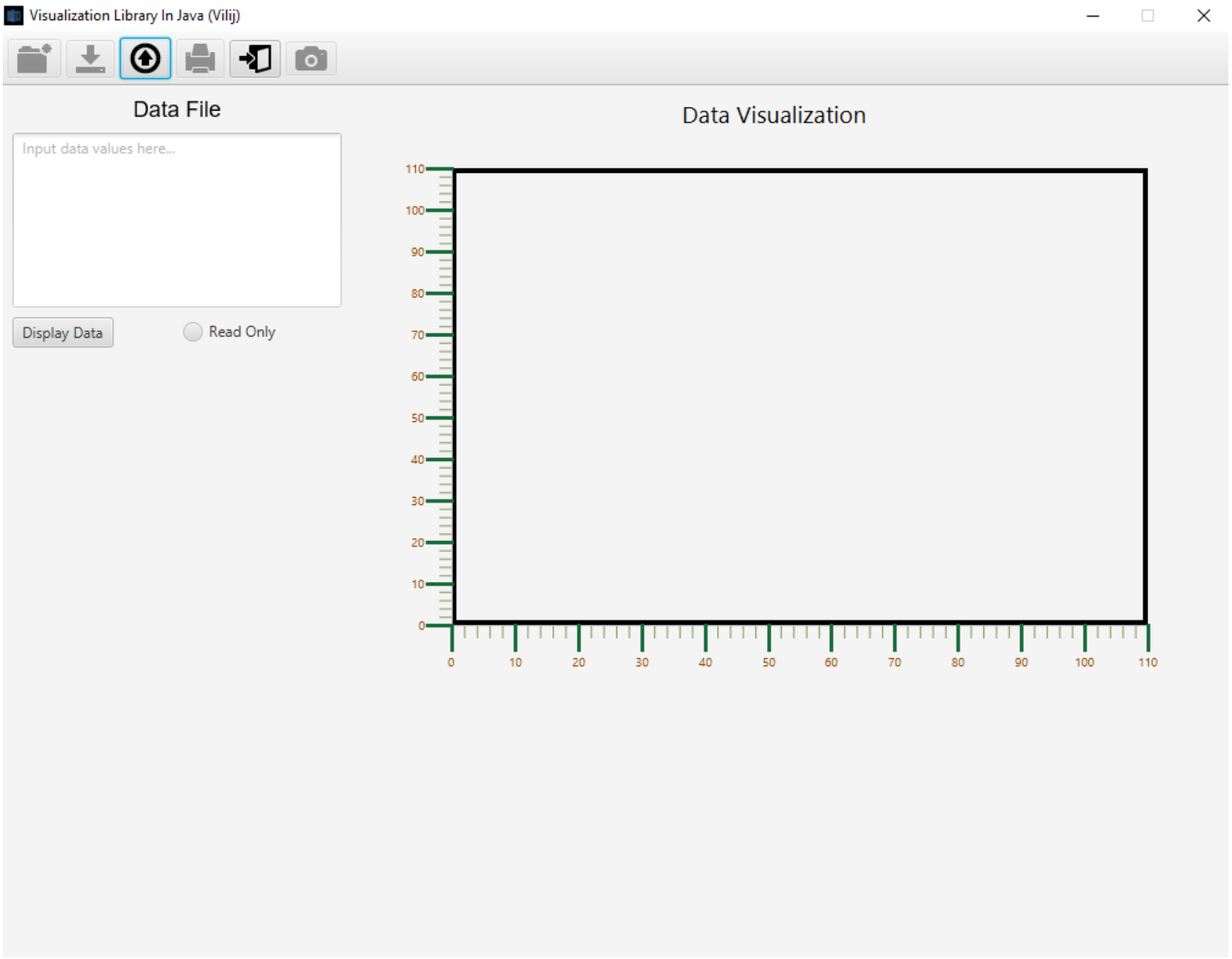

Figure 6.6: DATAVILIJ application user interface (UI) showcase as of Version 1.0.

In the above figure is the implementation of the DATAVILIJ application as of Version 1.0 (Homework 2) and includes a functional user interface where a user can type in data following the tab-separated (TSD) format and display data on the graph in terms of a scatter plot. For more information about the TSD format, look below in the appendix, specifically section 7.1.

The new data, load data, and save data functionalities are fully working and with error checking for the TSD format implemented as well.

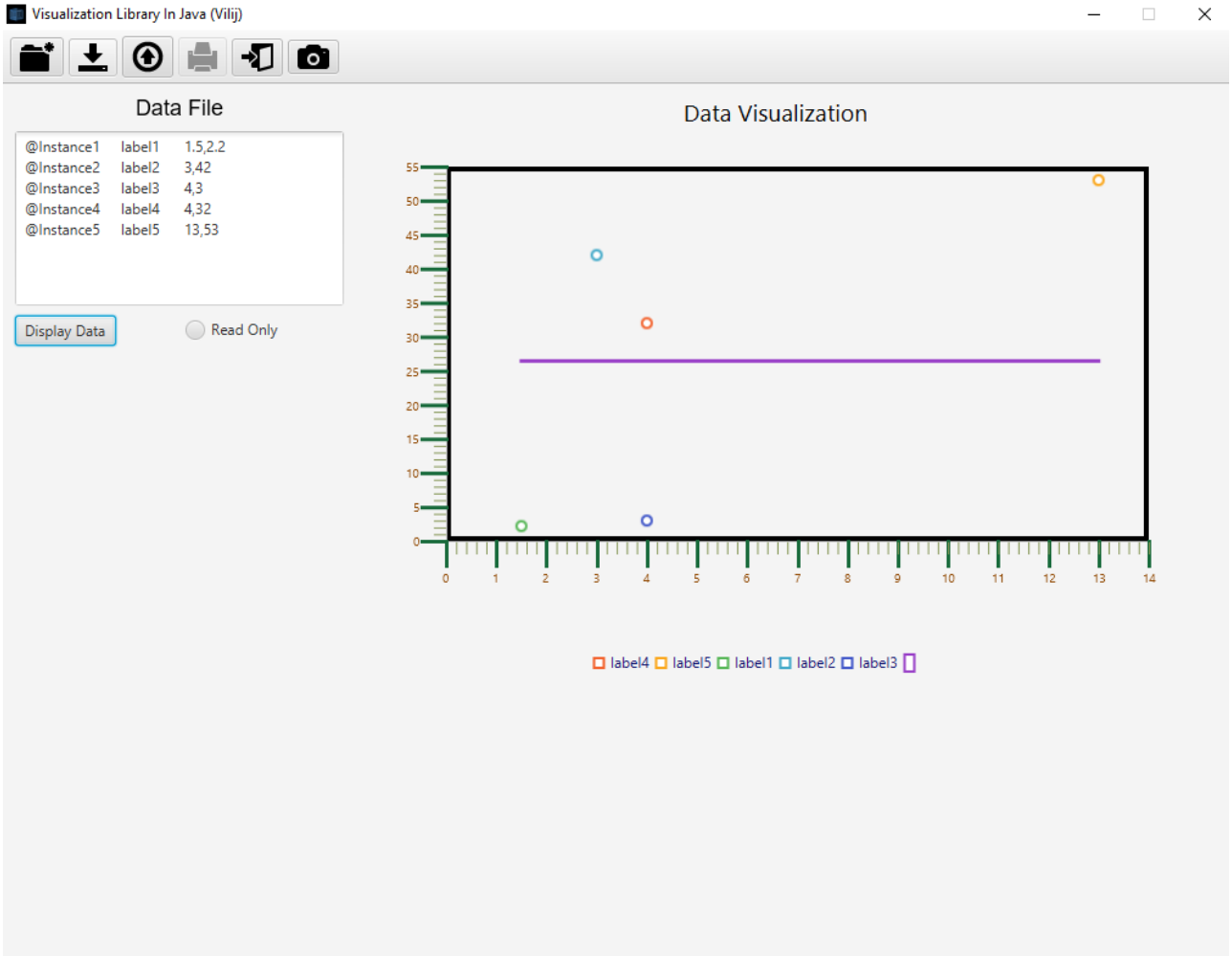

Figure 6.7: DATAVILIJ application user interface (UI) sample run.

In the above figure is a sample run of the DATAVILIJ application with some data loaded into the text area. The points are displayed on the graph with different colors due to the different labels used in the TSD format. An average line displaying the average of the y-values of each point in the data set is also correctly drawn through the data set. Tooltips are also placed, and will display the entire TSD format (instance name, label, coordinate) of the data point when hovered over each data point.

7 Supporting Information

This document serves as a reference when implementing the DATAVILIJ application and must be looked at as a guideline for how the application works and what was used to create it. Since this

program involves the use of a specific data format and specialized algorithms, here is provided the necessary information in this section in the form of three appendixes.

7.1 Appendix 1: 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.

7.2 Appendix 2: Characteristics of Learning Algorithms

The algorithms and their implementation, or an understanding of their internal workings, are not within the scope of this software. For the design and development, it suffices to understand some basic characteristics of these algorithms. There are provided in this appendix.

A. Classification Algorithms

These algorithms will try to categorize data into one of two known categories. The categories are the labels provided in the input data. The algorithm attempts to *learn* by trying to draw a straight line through the x-y 2-dimensional plane that separates the two labels as best as it can. Of course, this separation may not always be possible, but the algorithm tries nevertheless.

The output of a classification algorithm is thus a straight line. *This straight line is what is updated iteratively by the algorithm, and must be shown as part of the dynamically updating visualization in the GUI.*

Moreover, a classification algorithm will NOT change:

- the label of any instance provided as part of its input
- the actual *(x, y)* position of any instance in its input.

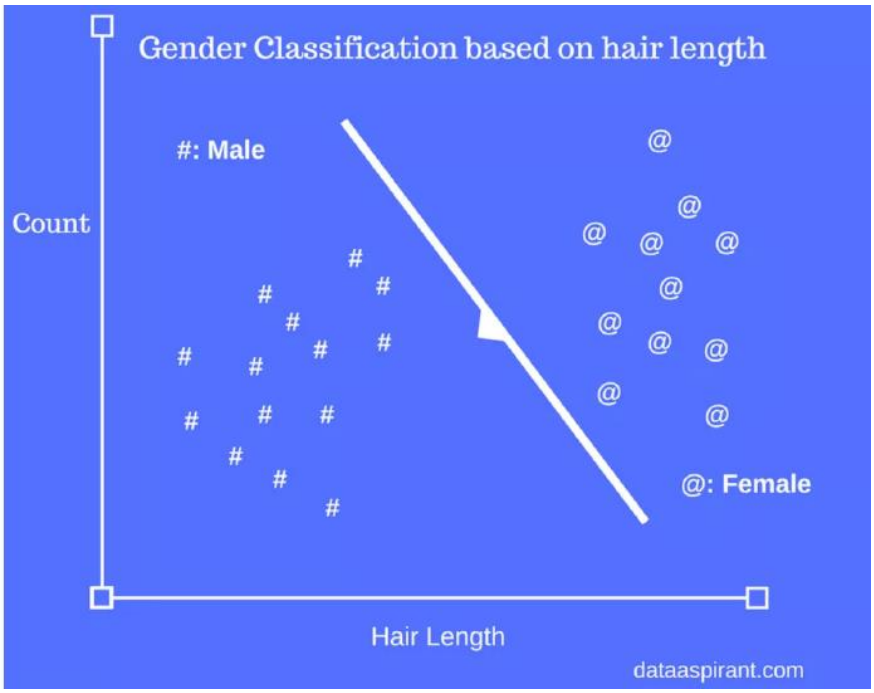

Figure 7.1: Output of a classification algorithm

An overly simplistic example where an algorithm is trying to separate two genders based on hair (x-axis) and count (y-axis) is shown in Figure 7.1 (courtesy of dataaspirant.com)

The run configuration of a classification algorithm should comprise the maximum number of iterations (it may, of course, stop before reaching this upper limit), the update interval, and the continuous run flag.

B. Clustering Algorithms

Clustering algorithms figure out patterns simply based on how data is distributed in space. These algorithms do not need any labels from the input data. Even if the input data has labeled instances, these algorithms will simply ignore them. They do, however, need to know the total number of labels ahead of time (i.e., before they start running). Therefore, their run configuration will comprise the maximum number of iterations (it may, just like classification algorithms, stop before reaching this upper limit), the update interval, the number of labels, and the continuous run flag.

The output of a clustering algorithm is, thus, the input instances, but with its own labels. *The instance labels get updated iteratively by the algorithm, and must be shown as part of the dynamically updating visualization in the GUI.*

This, along with the importance of providing the number of labels in the run configuration, is illustrated in Fig. 7.2 (a) and (b) below.

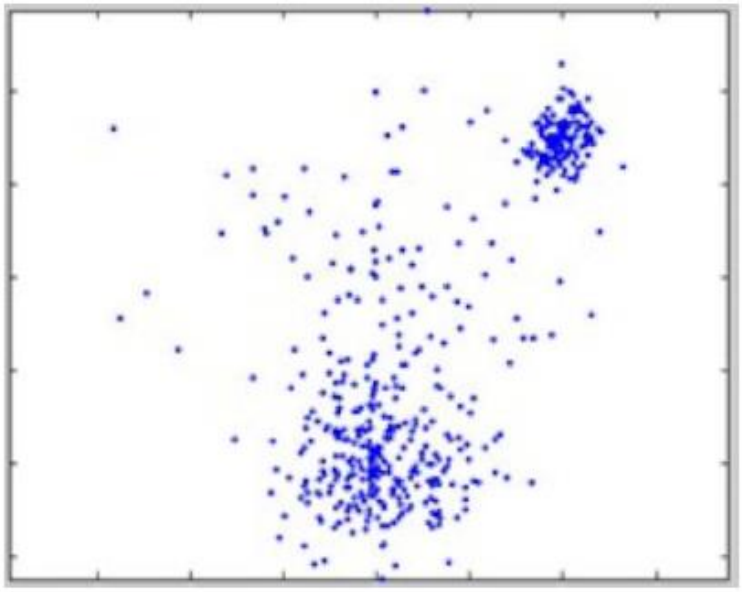

Figure 7.2(a): The input data need not have any labels for a clustering algorithm. Shown here as all instances having the same color. The number of labels decides what the output will look like

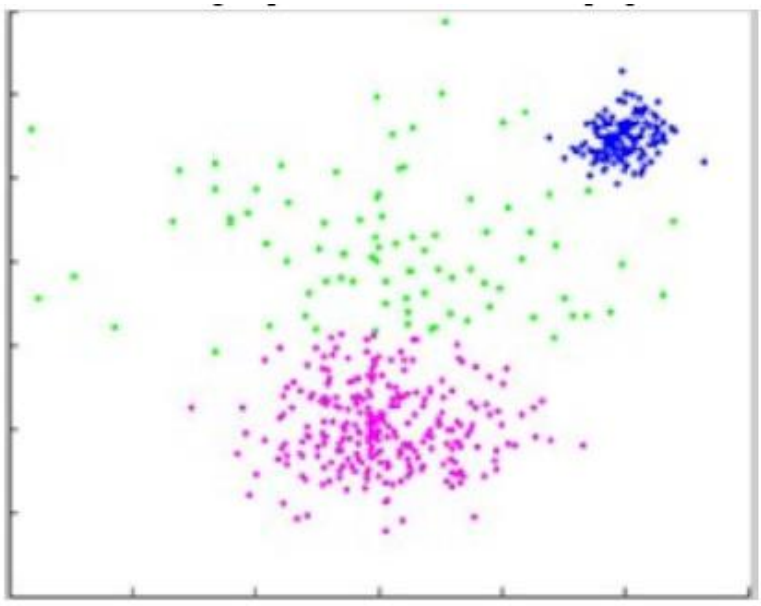

Figure 7.2(b): If the number of labels provided to the algorithm as part of its run configuration is 3, then this is what its output might look like. If the run configuration specified 2 labels, the green instances would get split between the blue and purple ones.

Just like a classification algorithm, **the actual position of any instance will not change**.

7.3 Appendix 3: Mock Algorithms for Testing

For testing, it suffices to create simple mock algorithms.

For classification algorithm testing, simply writing a test code that creates a random line every iteration is sufficient. To plot such a line in the GUI output, this algorithm can, for example, return a triple (a, b, c) of integers to represent the straight-line $ax + by + c = 0$, where a, b, and c are randomly generated using standard Java classes like java.util.Random.

For clustering algorithm testing, simply writing a test code that takes as input the number of labels (say, n) along with the data, and randomly assigns one of the labels $\{1, 2, \ldots, n\}$ to each instance.