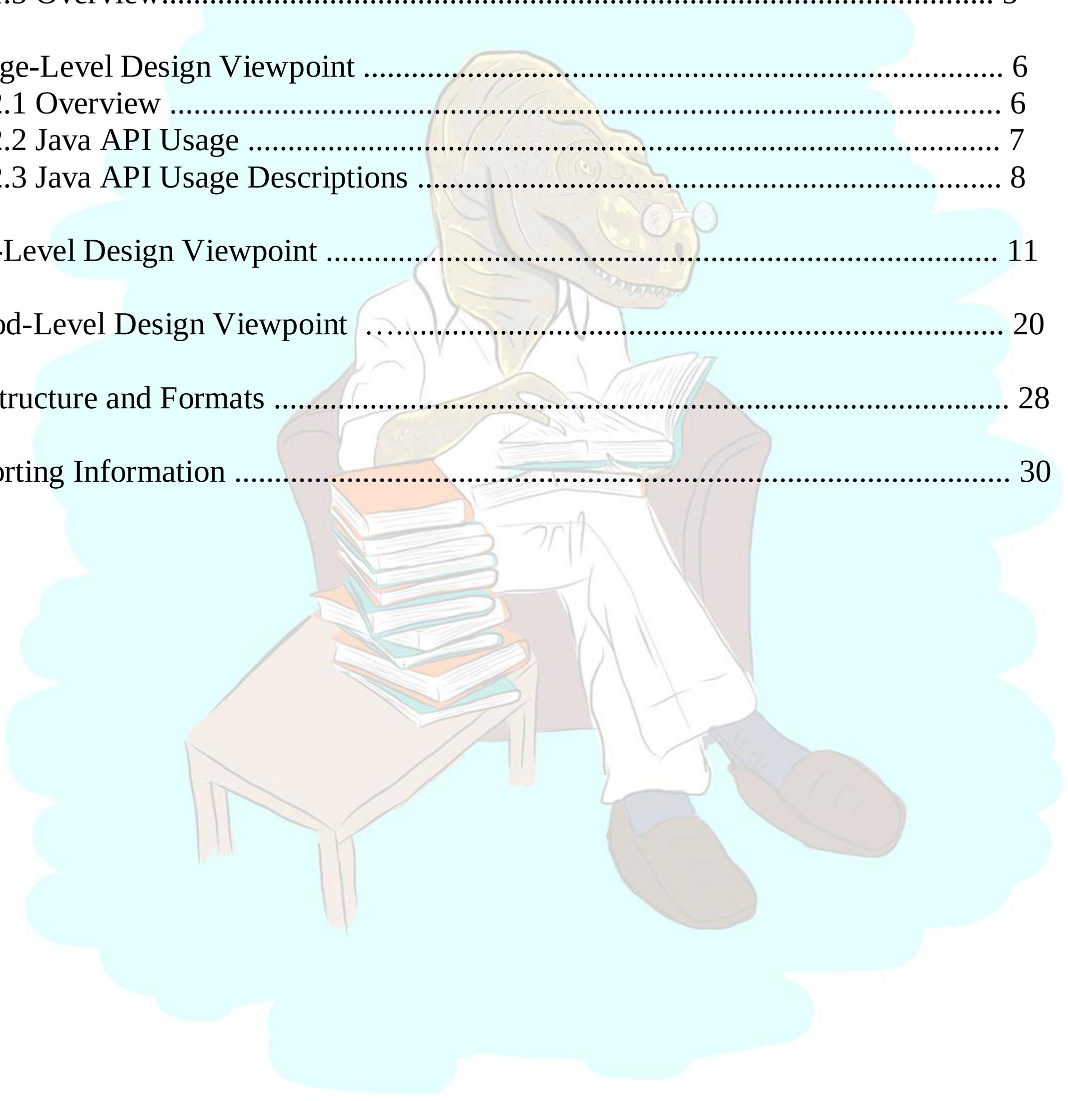
DATAVILIJ ™
Software Requirements Specification

DATAVILIJ ™

Author: Omar Zaraei
Professaur Inc.
March 2018
Version 1.0

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

Table of Contents



1. Introduction

Given the increasing importance of data-driven artificial intelligence (AI) in many aspects of computer science, visualizing how AI algorithms work is becoming increasingly important. Java is among the most important programming languages used to implement these algorithms, but it lacks standard data visualization libraries (unlike some other languages such as Python). Moreover, all existing libraries are meant to show us the final output of the data science algorithms. They are not designed for visualizing the changes that happen while the algorithms are running and updating the data. In other words, the visualization libraries do not help us see how these algorithms learn from the data. DataViLiJ (Data Visualization Library in Java) will be a desktop application that will allow users to select an algorithm (from a set of standard AI algorithms) and dynamically show the user what changes, and how.

1.1 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.

This document will show the whole design of the application using UML diagrams to provide details for all packages, member variables and methods of classes. It will also show the visualization of how classes and objects interact with one another.

1.2 Intended Audience

The intended audience for this document is the development team, including instructors, software designer, and software developers.

1.3 Scope

The goal of this project is for students and beginning professionals in 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. DataViLiJ 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. As such, 3D visualization is currently beyond the scope of DataViLiJ.

1.4 Definitions, acronyms, and abbreviations

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.

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

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.

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.

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.

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.

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.

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.

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.

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

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.

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

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

1.5 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 Engineering 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.

1.6 Overview

This software requirements specification (SRS) document will clearly define the operational capabilities of DataViLiJ and its UI functionalities and aesthetics. Note that this document is not a software design description (SDD), and it does not include design components that use UML or specify how to build the appropriate technologies. It is simply an agreement concerning what to build. The remainder of this document consists of two chapters, appendices, and the index. Chapter 2 provides the general factors affecting the application requirements. It also provides the background for these requirements, putting the application in perspective with other related applications such as the algorithms that will be used. Chapter 3 provides the software requirements to a level of detail sufficient to allow the design of the application to satisfy those requirements. This includes the GUI and the core operations. Here, every input into the application and every output from the application, along with all the functions, are described. The appendices include sample data that illustrates the data format, and background information on the type of algorithms.

2. Package-Level Design Viewpoint

The DataViLiJ application, will be built using the vilij Framework, xmlutil Framework.
The application will heavily rely on the JAVA API to be built. The following is the class package level design of the application.

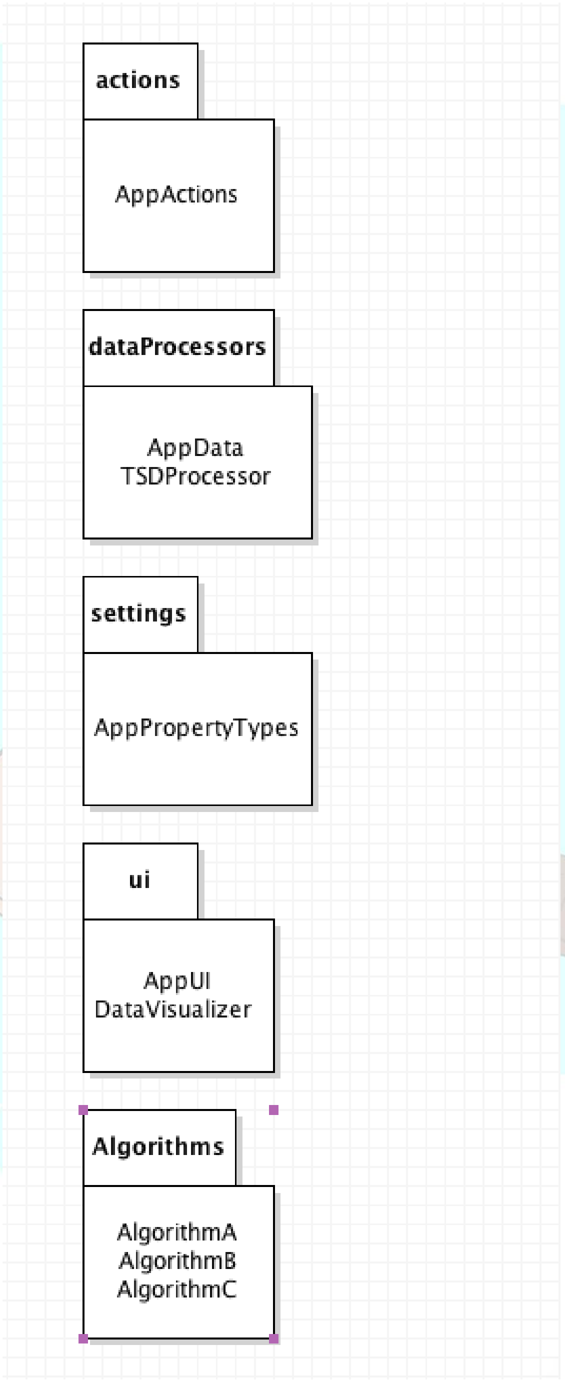


Figure 2.1 specifies all the classes to be built and placed in their respectful packages

2.2 Java API Usage

All the frameworks and the DataViLiJ application will be developed using JAVA 8. The following is all the use of the JAVA API.

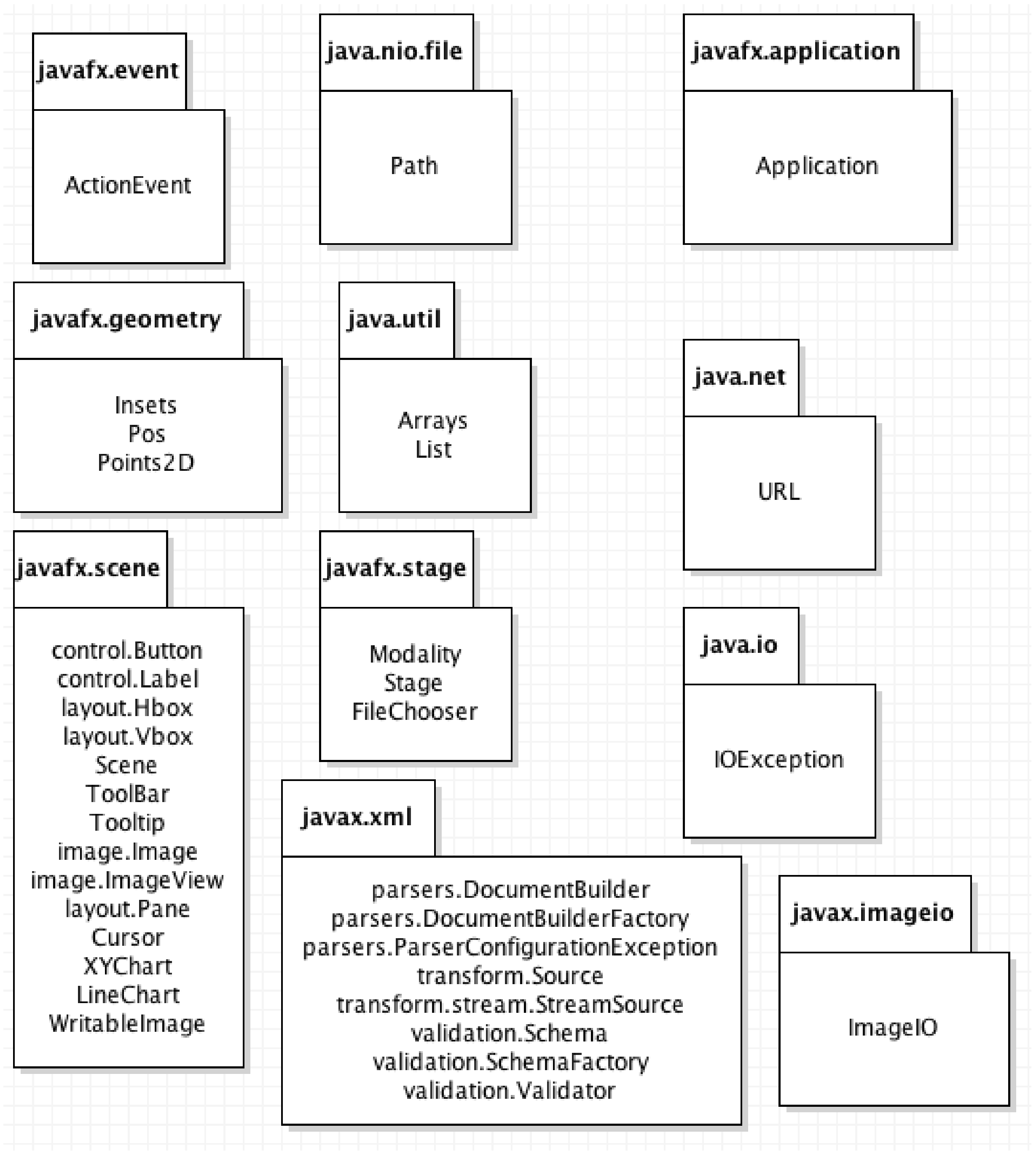


Figure 2.2 shows all the JAVA built-in API that will be used.

2.3 Java API Usage Descriptions

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

Class/Interface	Use
Application	For our application to run the JavaFX Application class.

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

Class/Interface Use	Use
Stage	To create the main window of the GUI.
FileChooser	To be used for providing the options to save, load and open files.
Modality	To be used for the different types of Stage.

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

Class/Interface	Use
Label	The address of the point in the graph which is located inside the GUI.
Scene	A Scene is contained inside A Stage.
ToolBar	ToolBar contains the buttons which help user navigate.
VBox	To make vertical windows in the GUI.
HBox	To make horizontal windows in the GUI.
layoutPane	To expose the children in the GUI.
Tooltip	When user hovers over a point in the graph, a tooltip will provide information about the point such as name, value, etc.
Button	To help the user navigate through the application.
Cursor	When the default cursor moves into the graph, the cursor will change from default cursor to another cursor.
WritableImage	When the user screenshots a graph, we need to save it. A WritableImage helps us do that.
LineChart	A LineChart is used to show the line of average on the graph.

Image	For showing and loading images.
ImageView	For editing images.

Table 2.4: Uses for classes in the Java API's javaio package

Class/Interface	Use
IO Exception	An IO Exception will be thrown if there is an error of any sort while loading or saving a file.

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

Class/Interface	Use
Points2D	To create points on a graph.
Insets	Used for padding inside of a GUI Stage.
Pos	A set of values describing vertical and horizontal positioning and alignment.

Table 2.6: Uses for classes in the Java API's javafx.Imageio package

Class/Interface	Use
ImageIO	Used for loading/reloading an image.

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

Class/Interface	Use
Path	The path where a file can be saved or opened from.

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

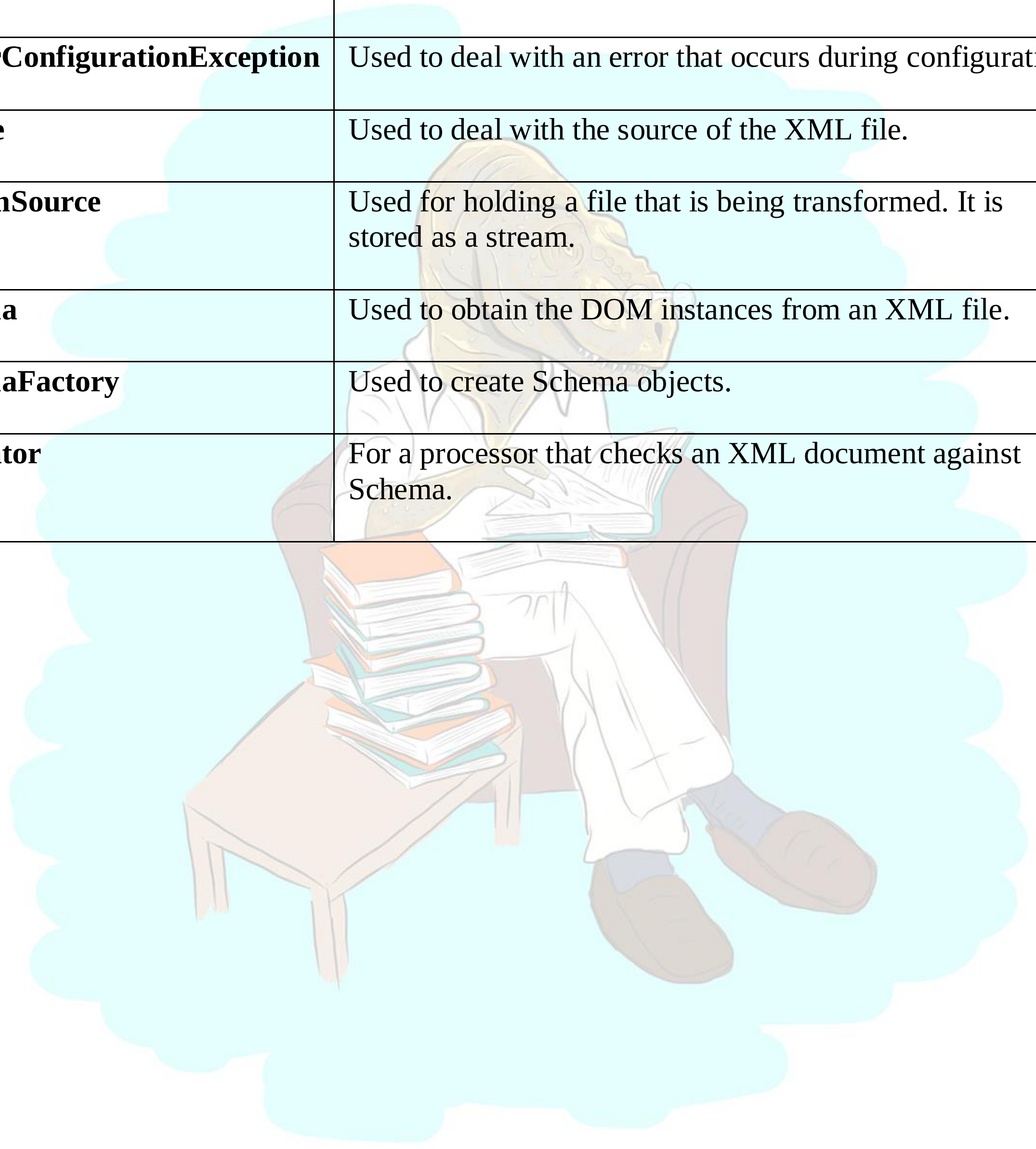
Class/Interface	Use
URL	Address of a website.

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

Class/Interface	Use
ArrayList	An implementation of a resizable array. We will use this in multiple places to store data.
HashMap	A data structure that holds data in a specific order.

Table 2.10: Uses for classes in the Java API's javax.xml package

Class/Interface	Use
DocumentBuilder	Used to define the API to obtain DOM instances from a document.
DocumentBuilderFactory	Used for obtaining a parser that produces DOM object trees from XML files.
ParserConfigurationException	Used to deal with an error that occurs during configuration.
Source	Used to deal with the source of the XML file.
StreamSource	Used for holding a file that is being transformed. It is stored as a stream.
Schema	Used to obtain the DOM instances from an XML file.
SchemaFactory	Used to create Schema objects.
Validator	For a processor that checks an XML document against Schema.



3. Class-Level Design Viewpoint

The DataViLiJ application will use the Vilij Framework and xmlutil Framework. The UML diagram below shows us all the relationships between the classes. The UML diagrams after this will go more in depth with the relationship between the classes.

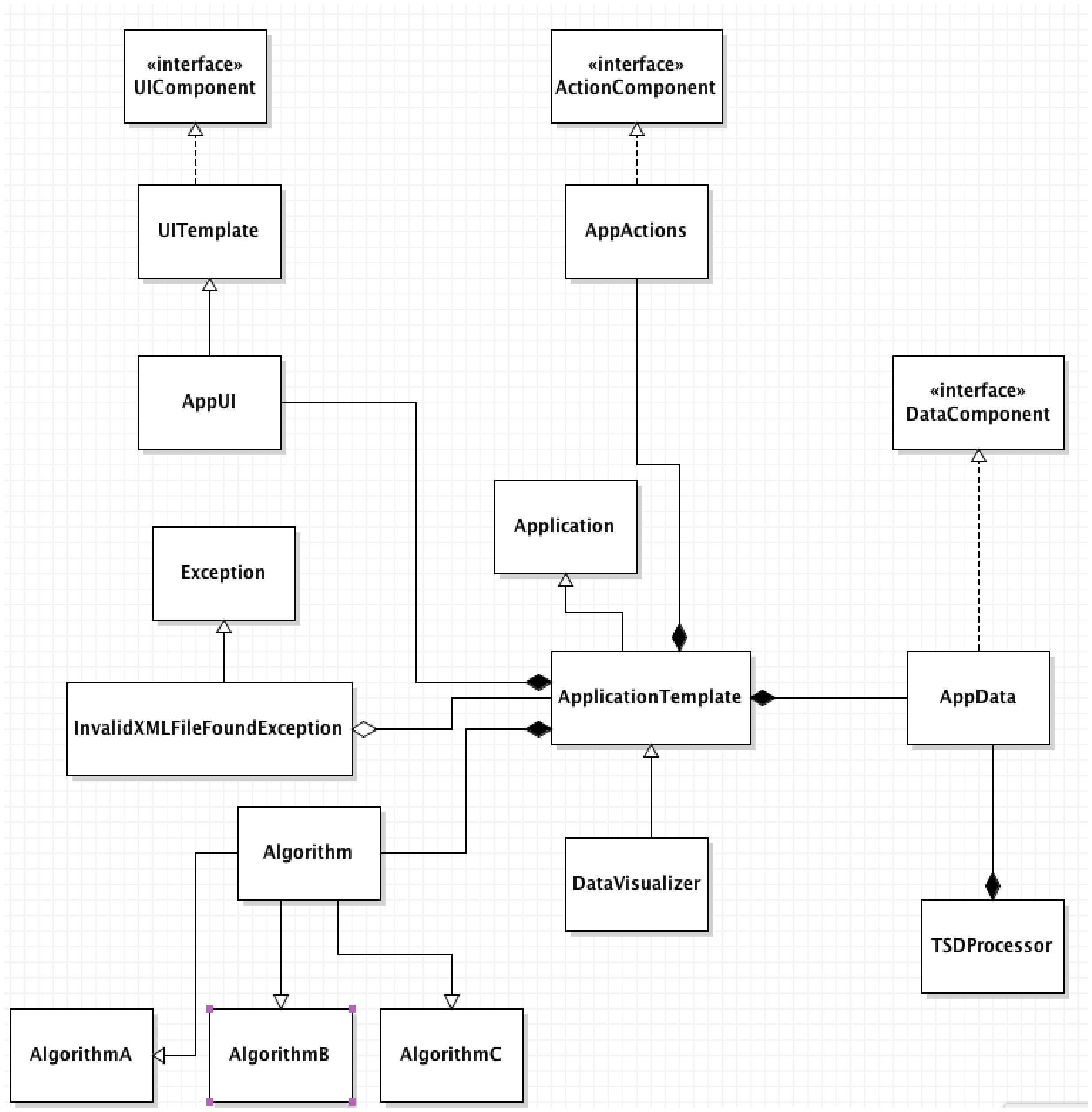


Figure 3.1: UML Class Diagram for DataViLiJ Application.

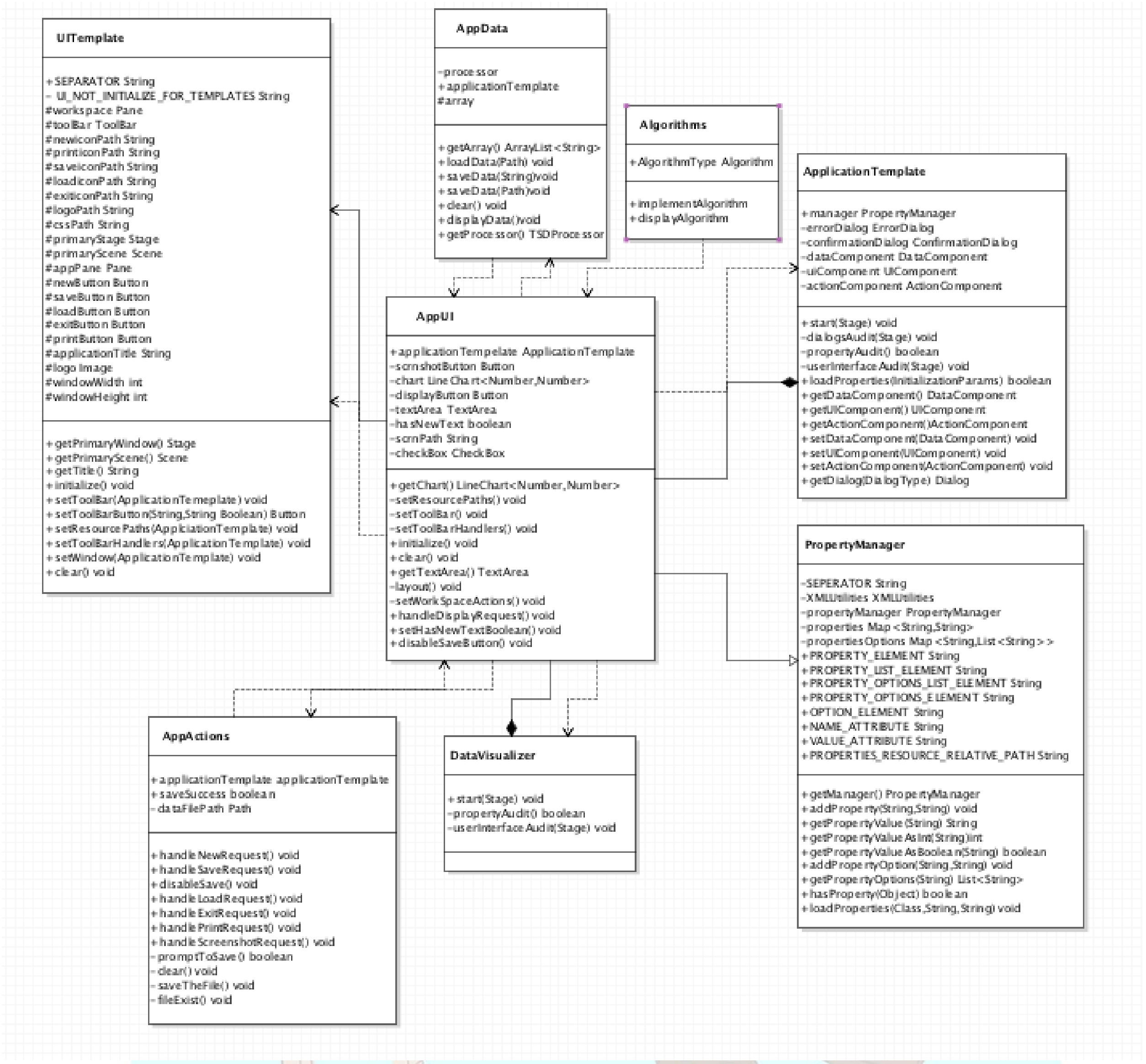

UITemplate
+SEPARATOR String
- UI_NOT_INITIALIZE_FOR_TEMPLATES String
#workspace Pane
#toolBar ToolBar
#newiconPath String
#printiconPath String
#saveiconPath String
#loadiconPath String
#exiticonPath String
#logoPath String
#cssPath String
#primaryStage Stage
#primaryScene Scene
#appPane Pane
#newButton Button
#saveButton Button
#loadButton Button
#exitButton Button
#printButton Button
#applicationTitle String
#logo Image
#windowWidth int
#windowHeight int
+getPrimaryWindow() Stage
+getPrimaryScene() Scene
+getTitle() String
+initialize() void
+setToolBar(ApplicationTemeplate) void
+setToolBarButton(String,String Boolean) Button
+setResourcePaths(AppliciationTemplate) void
+setToolBarHandlers(ApplicationTemplate) void
+setWindow(ApplicationTemplate) void
+clear() void
AppData
-processor
+applicationTemplate
#array
+getArray() ArrayList<String>
+loadData(Path) void
+saveData(String)void
+saveData(Path)void
+clear() void
+displayData()void
+getProcessor() TSDProcessor
Algorithms
+AlgorithmType Algorithm
+implementAlgorithm
+displayAlgorithm
ApplicationTemplate
+manager PropertyManager
-errorDialog ErrorDialog
-confirmationDialog ConfirmationDialog
-dataComponent DataComponent
-uiComponent UIComponent
-actionComponent ActionComponent
+start(Stage) void
-dialogsAudit(Stage) void
-propertyAudit() boolean
-userInterfaceAudit(Stage) void
+loadProperties(InitializationParams) boolean
+getDataComponent() DataComponent
+getUIComponent() UIComponent
+getActionComponent()ActionComponent
+setDataComponent(DataComponent) void
+setUIComponent(UIComponent) void
+setActionComponent(ActionComponent) void
+getDialog(DialogType) Dialog
AppUI
+applicationTempelate ApplicationTemplate
-scrnshotButton Button
-chart LineChart<Number,Number>
-displayButton Button
-textArea TextArea
-hasNewText boolean
-scrnPath String
-checkBox CheckBox
+getChart() LineChart<Number,Number>
-setResourcePaths() void
-setToolBar() void
-setToolBarHandlers() void
+initialize() void
+clear() void
+getTextArea() TextArea
-layout() void
-setWorkSpaceActions() void
+handleDisplayRequest() void
+setHasNewTextBoolean() void
+disableSaveButton() void
PropertyManager
-SEPERATOR String
-XMLUtilities XMLUtilities
-propertyManager PropertyManager
-properties Map<String,String>
-propertiesOptions Map<String,List<String>>
+PROPERTY_ELEMENT String
+PROPERTY_LIST_ELEMENT String
+PROPERTY_OPTIONS_LIST_ELEMENT String
+PROPERTY_OPTIONS_ELEMENT String
+OPTION_ELEMENT String
+NAME_ATTRIBUTE String
+VALUE_ATTRIBUTE String
+PROPERTIES_RESOURCE_RELATIVE_PATH String
+getManager() PropertyManager
+addProperty(String,String) void
+getPropertyValue(String) String
+getPropertyValueAsInt(String)int
+getPropertyValueAsBoolean(String) boolean
+addPropertyOption(String,String) void
+getPropertyOptions(String) List<String>
+hasProperty(Object) boolean
+loadProperties(Class,String,String) void
AppActions
+applicationTemplate applicationTemplate
+saveSuccess boolean
-dataFilePath Path
+handleNewRequest() void
+handleSaveRequest() void
+disableSave() void
+handleLoadRequest() void
+handleExitRequest() void
+handlePrintRequest() void
+handleScreenshotRequest() void
-promptToSave() boolean
-clear() void
-saveTheFile() void
-fileExist() void
DataVisualizer
+start(Stage) void
-propertyAudit() boolean
-userInterfaceAudit(Stage) void

Figure 3.2: UML Class Diagram for Relationship between classes

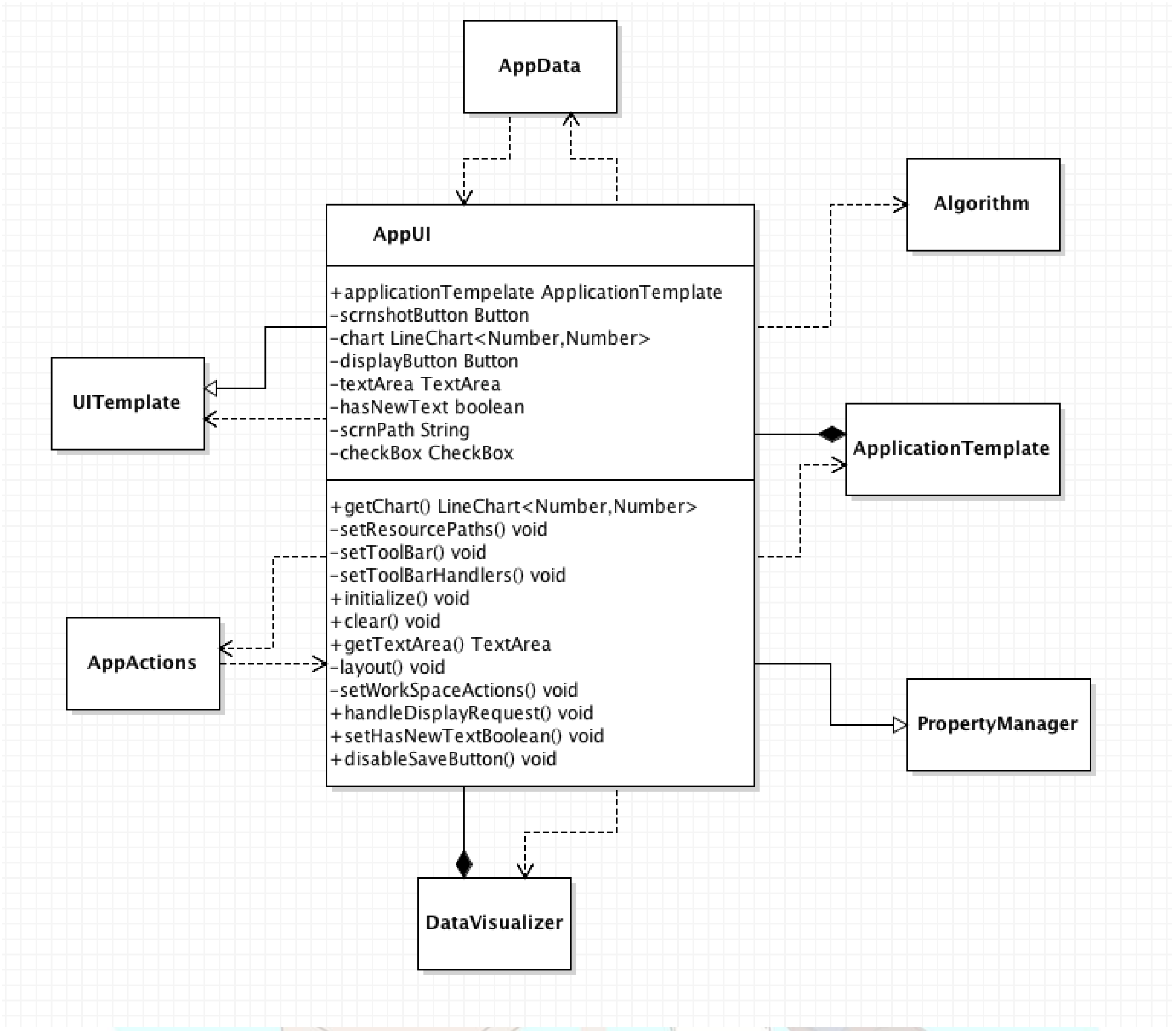


Figure 3.3: UML Diagram for the relationship of AppUI with other classes

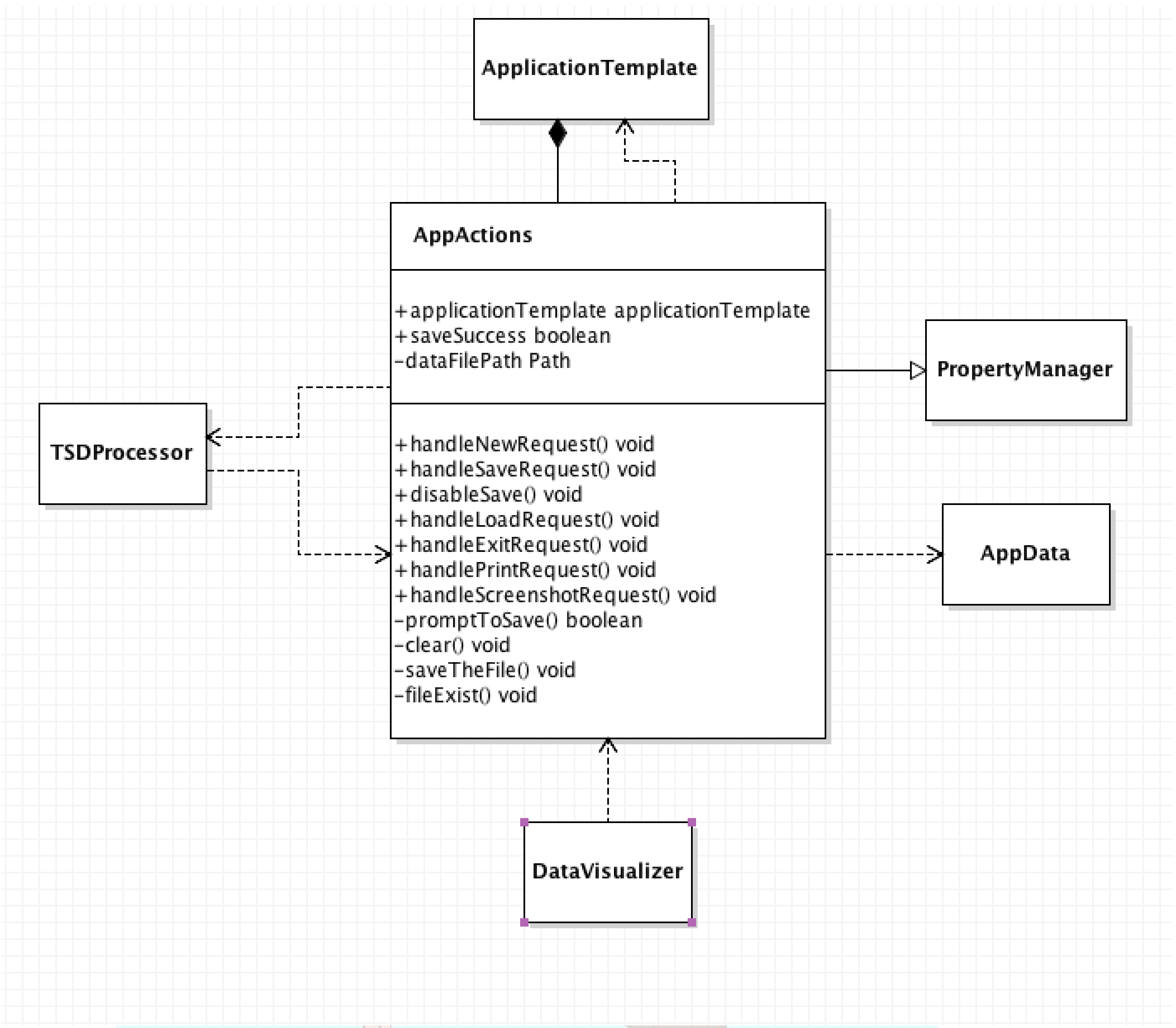


Figure 3.4: UML Diagram for the relationship of AppActions with other classes

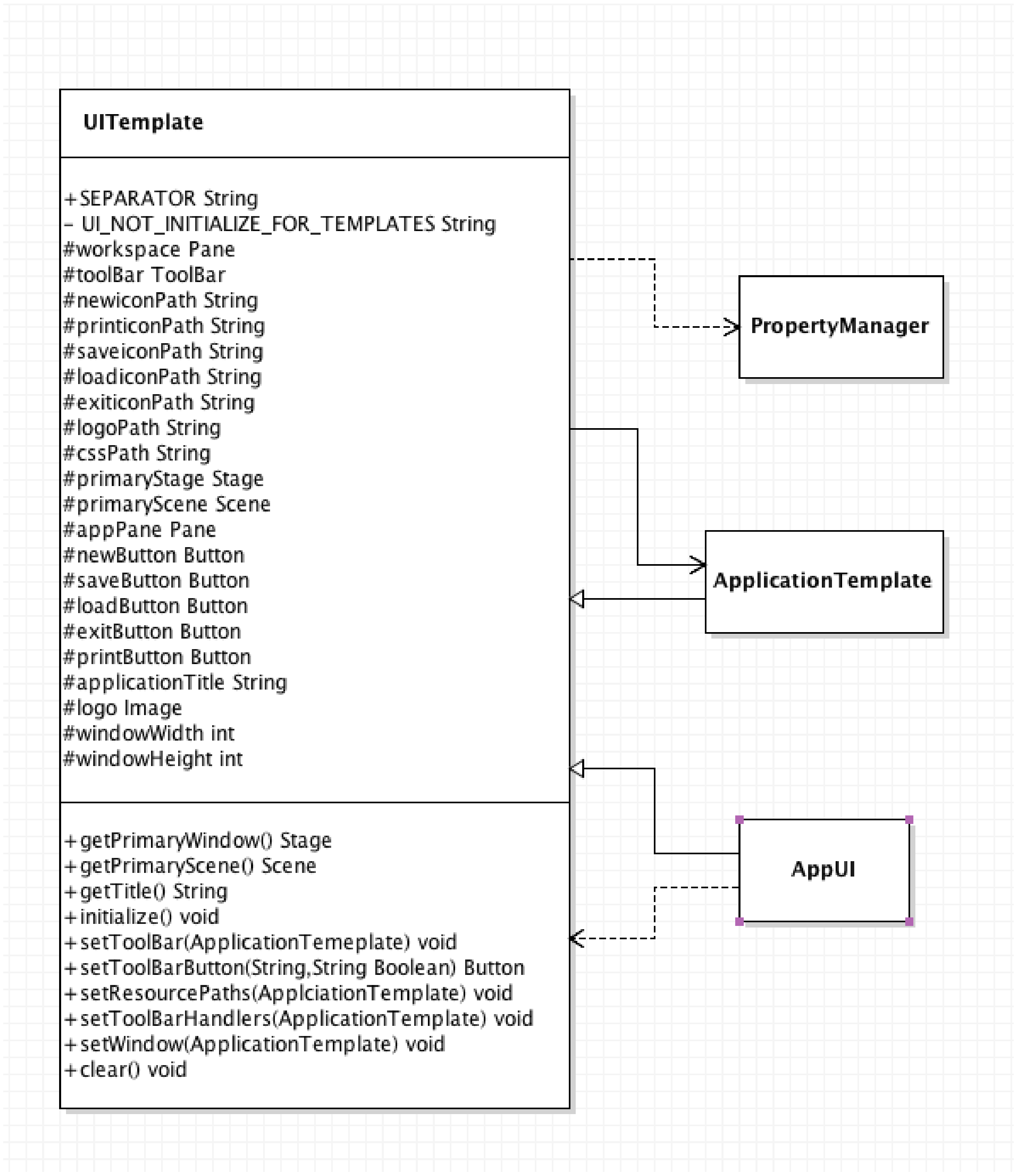


Figure 3.5: UML Diagram for the relationship of UITemplate with other classes

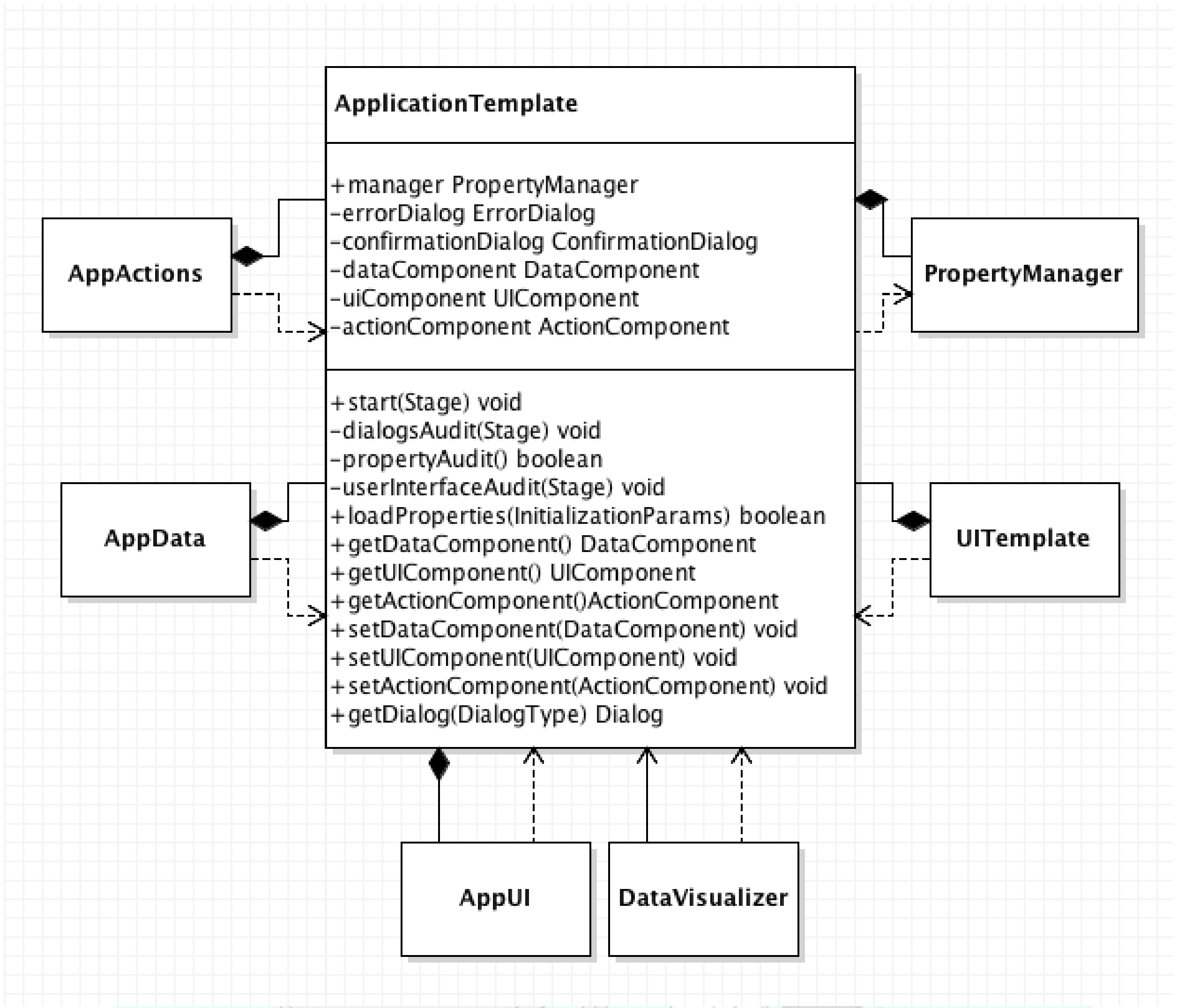

ApplicationTemplate
+manager PropertyManager
-errorDialog ErrorDialog
-confirmationDialog ConfirmationDialog
-dataComponent DataComponent
-uiComponent UIComponent
-actionComponent ActionComponent
+start(Stage) void
-dialogsAudit(Stage) void
-propertyAudit() boolean
-userInterfaceAudit(Stage) void
+loadProperties(InitializationParams) boolean
+getDataComponent() DataComponent
+getUIComponent() UIComponent
+getActionComponent()ActionComponent
+setDataComponent(DataComponent) void
+setUIComponent(UIComponent) void
+setActionComponent(ActionComponent) void
+getDialog(DialogType) Dialog
AppActions
PropertyManager
AppData
UITemplate
AppUI
DataVisualizer

Figure 3.6: UML Diagram for the relationship of ApplicationTemplate with other classes

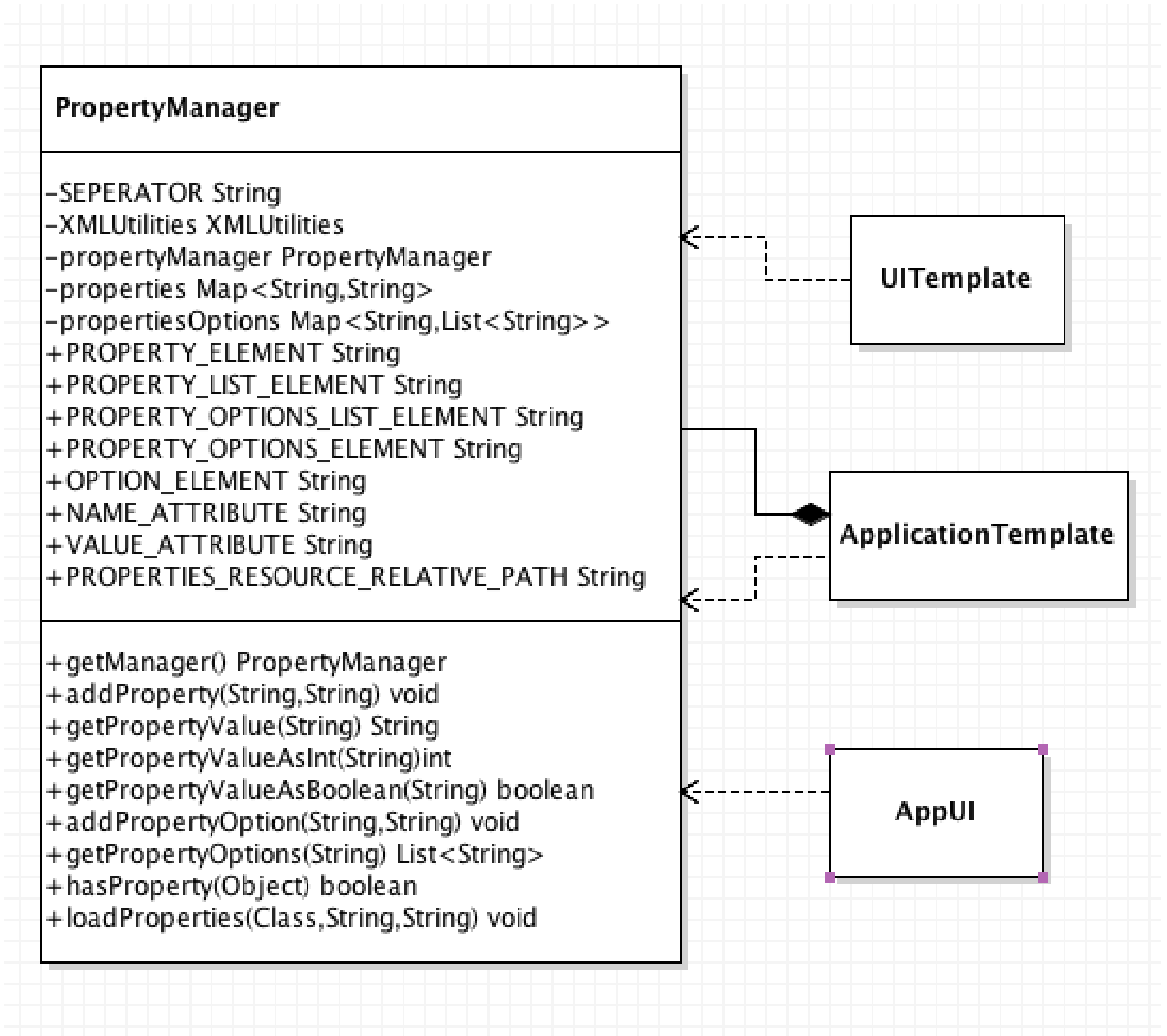

PropertyManager
-SEPERATOR String
-XMLUtilities XMLUtilities
-propertyManager PropertyManager
-properties Map<String,String>
-propertiesOptions Map<String,List<String>>
+PROPERTY_ELEMENT String
+PROPERTY_LIST_ELEMENT String
+PROPERTY_OPTIONS_LIST_ELEMENT String
+PROPERTY_OPTIONS_ELEMENT String
+OPTION_ELEMENT String
+NAME_ATTRIBUTE String
+VALUE_ATTRIBUTE String
+PROPERTIES_RESOURCE_RELATIVE_PATH String
+getManager() PropertyManager
+addProperty(String,String) void
+getPropertyValue(String) String
+getPropertyValueAsInt(String)int
+getPropertyValueAsBoolean(String) boolean
+addPropertyOption(String,String) void
+getPropertyOptions(String) List<String>
+hasProperty(Object) boolean
+loadProperties(Class,String,String) void
UITemplate
ApplicationTemplate
AppUI

Figure 3.7: UML Diagram for the relationship of PropertyManager with other classes

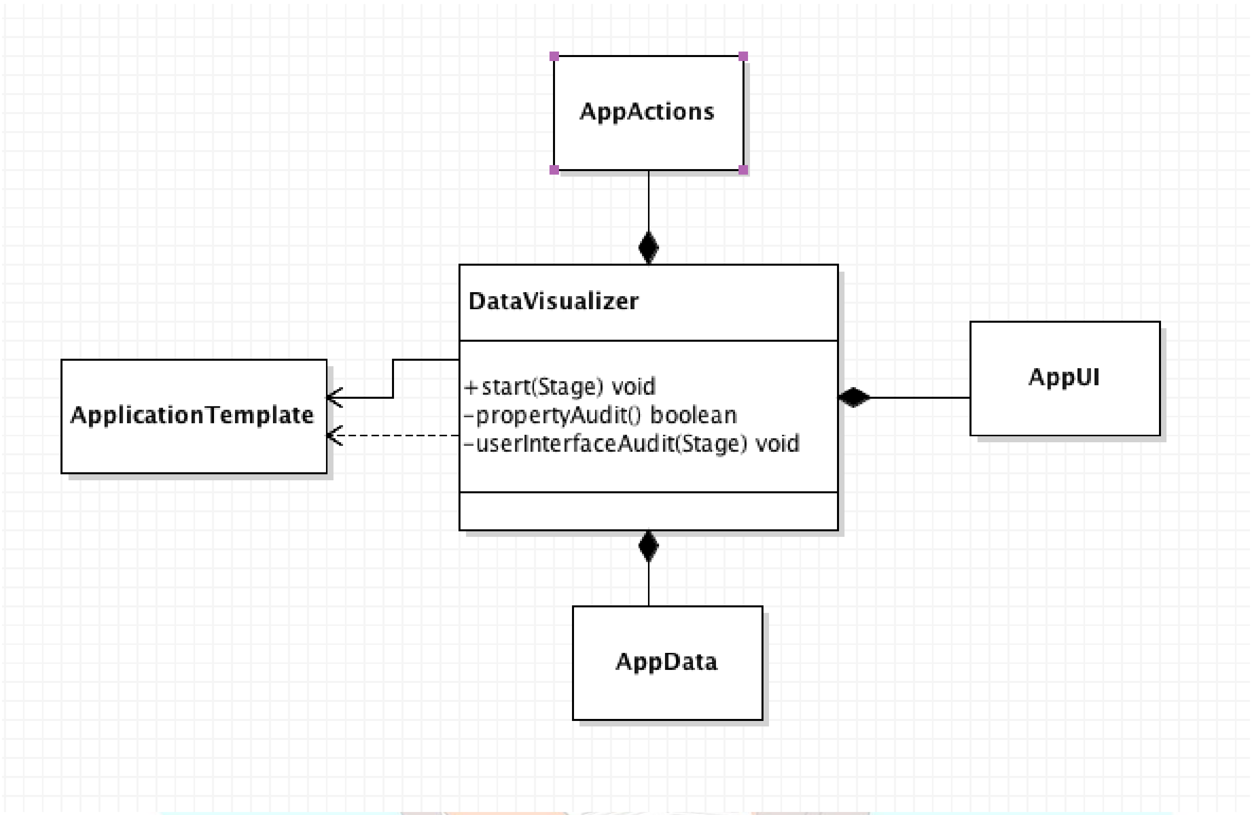
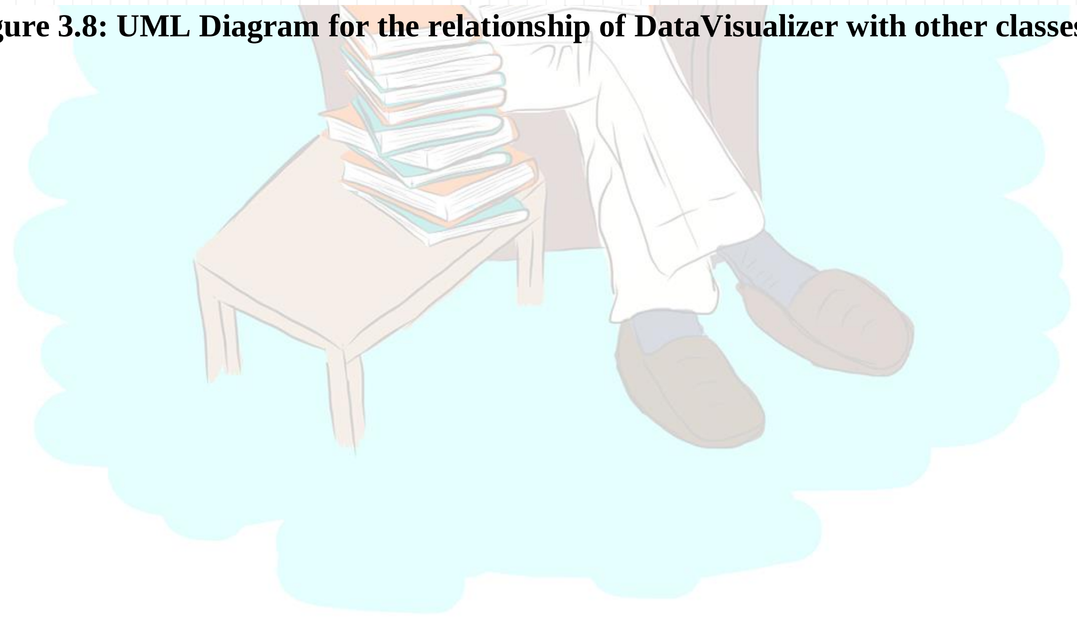


Figure 3.8: UML Diagram for the relationship of DataVisualizer with other classes



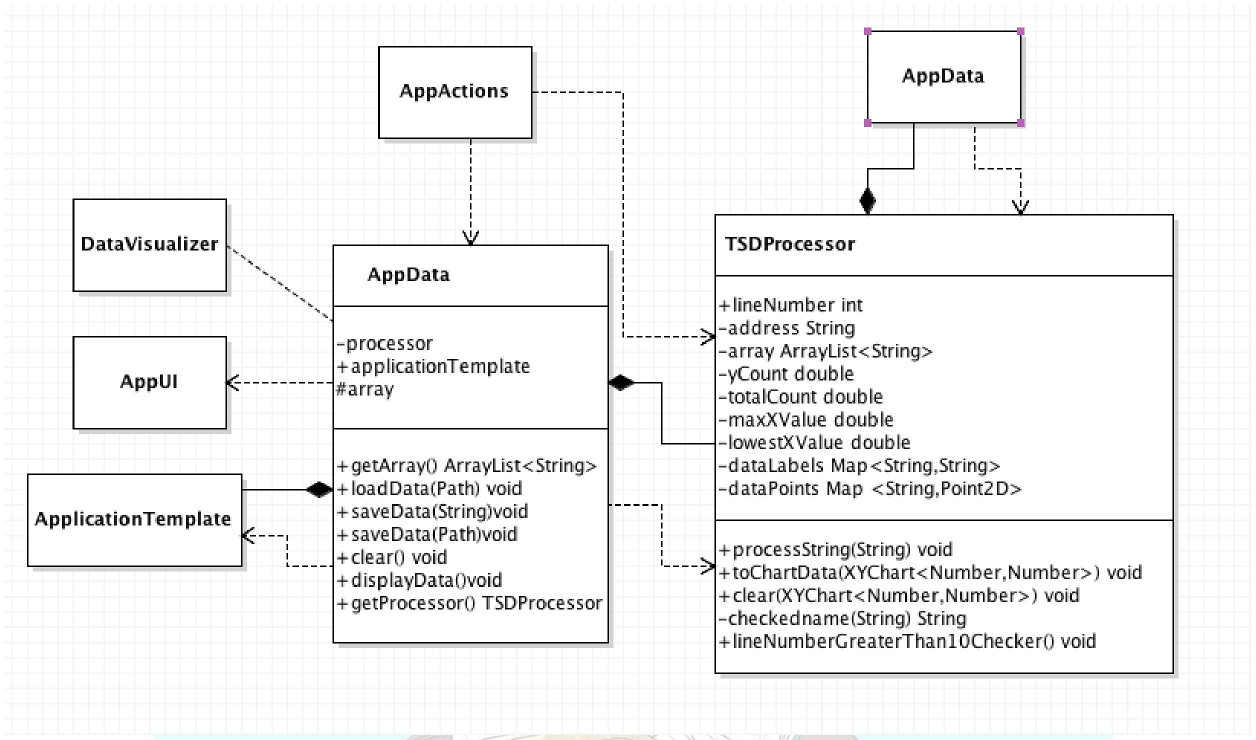


Figure 3.9: UML Diagram for the relationship of AppData and TSDProcessor with other classes

4. Method-Level Design Viewpoint

We have designed how all the classes will be implemented and their relationship with each other. Now we will show how the program operates. The flow of the program will be showed in the next couple of UML Sequence Diagrams.

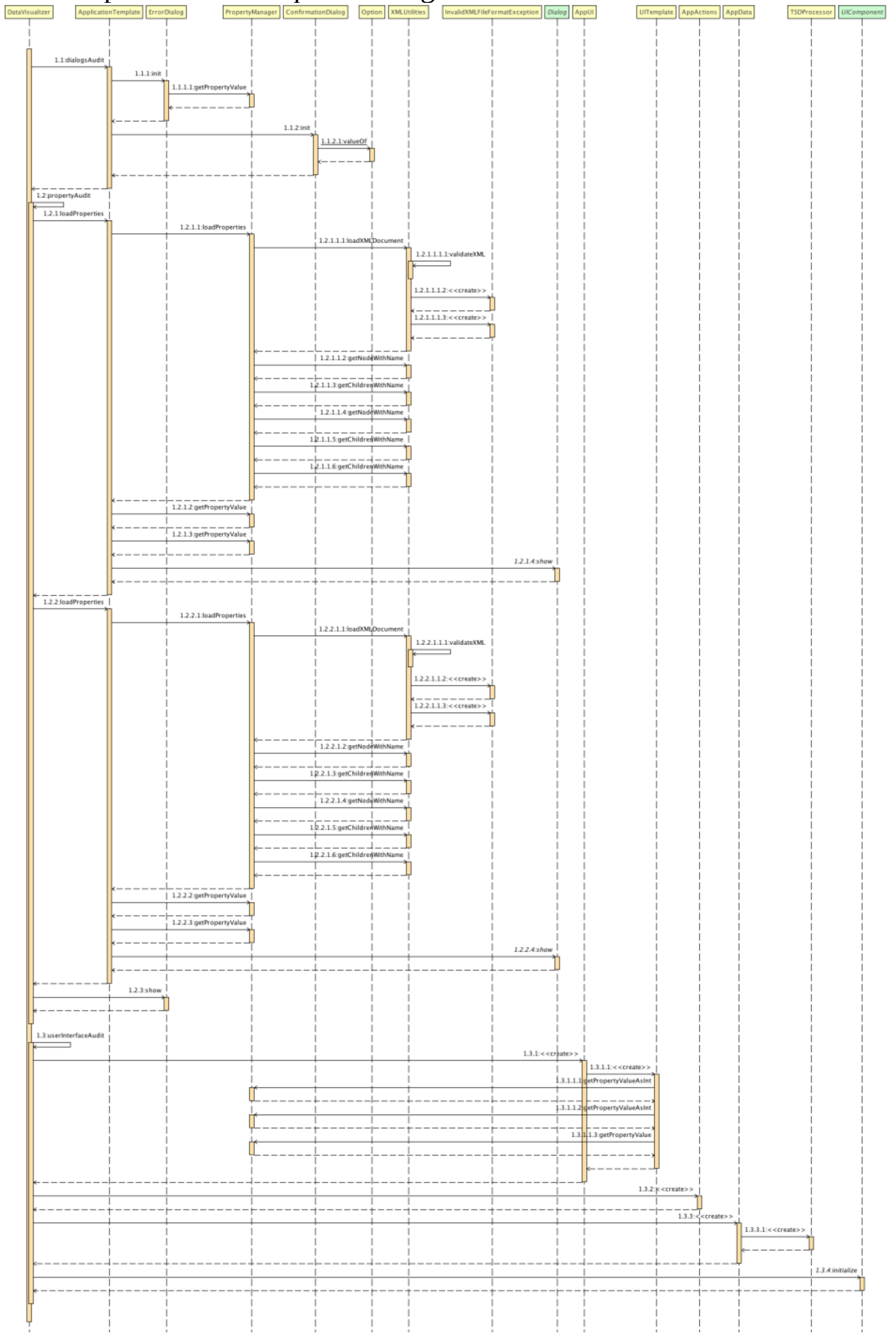


Figure 4.1 Start Application Use UML Sequence Diagram

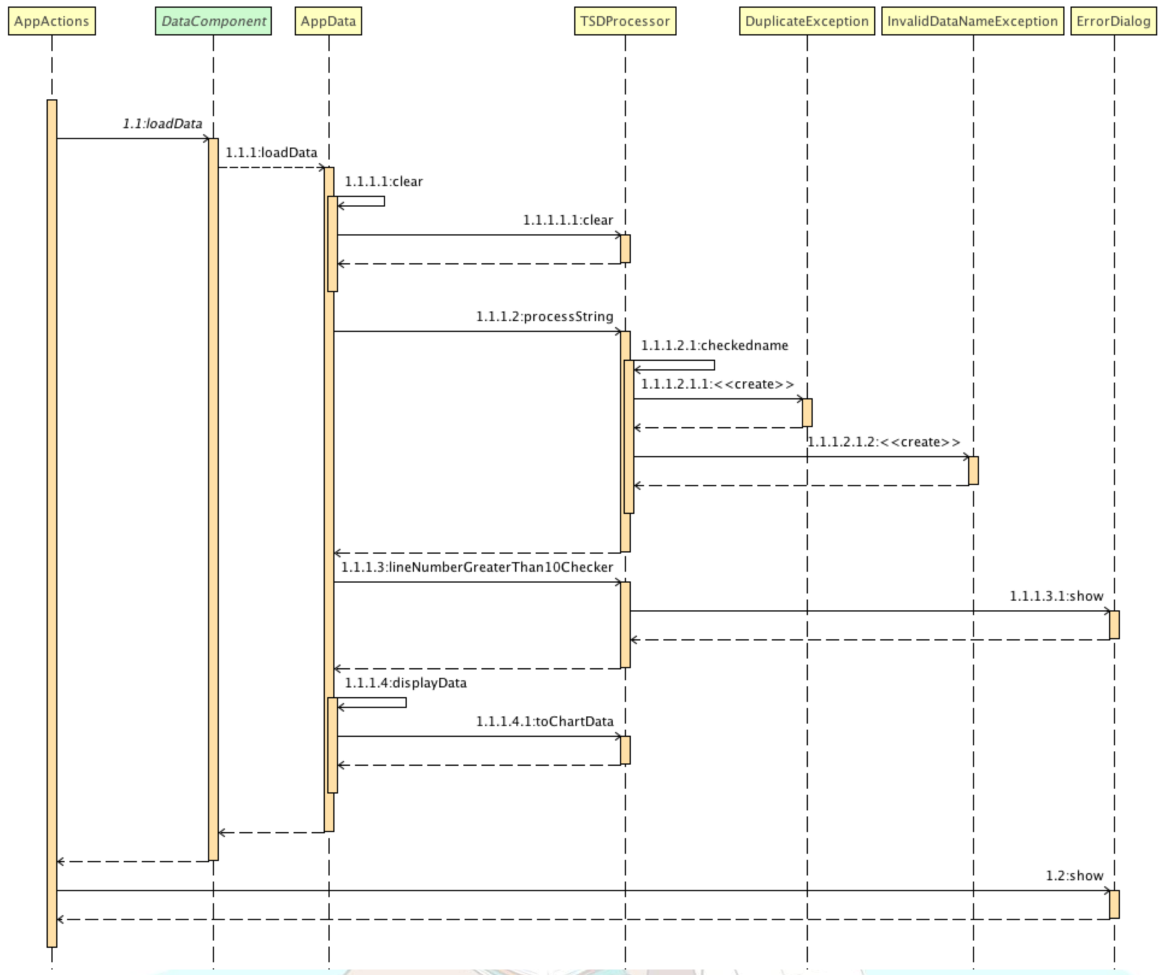
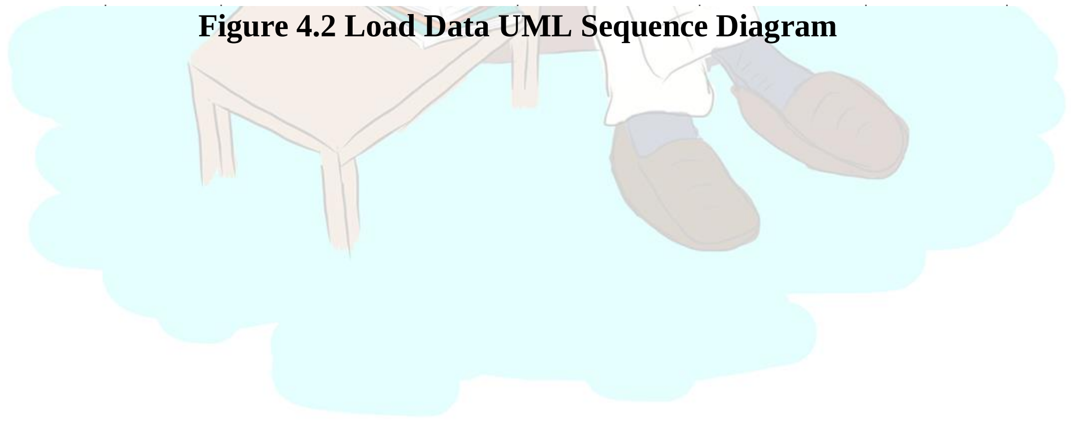


Figure 4.2 Load Data UML Sequence Diagram



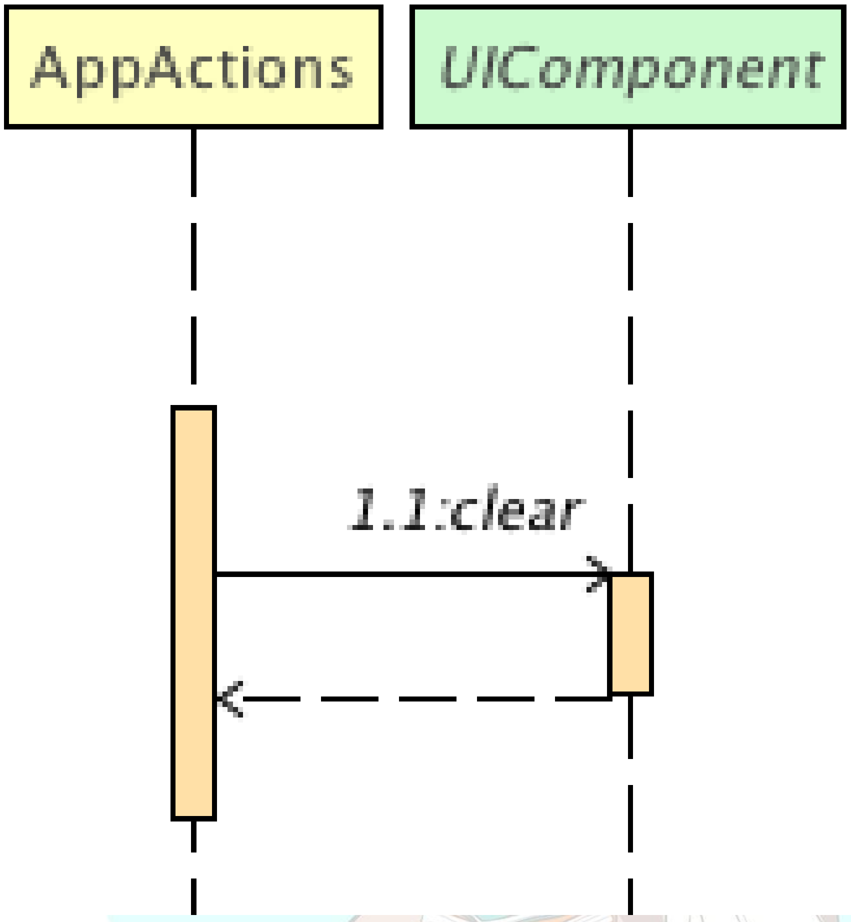


Figure 4.3 New Data UML Sequence Diagram

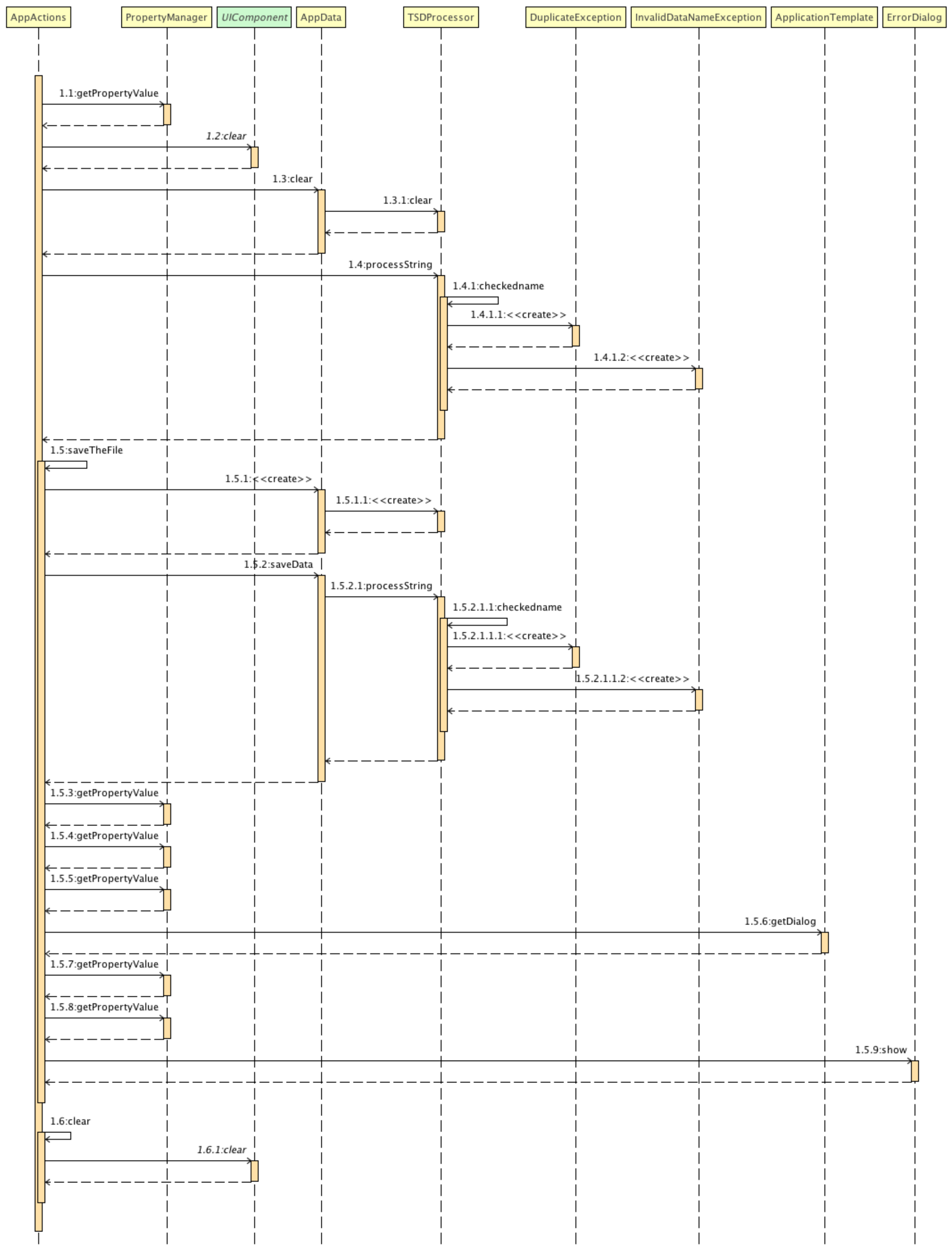


Figure 4.4 Save Data UML Sequence Diagram

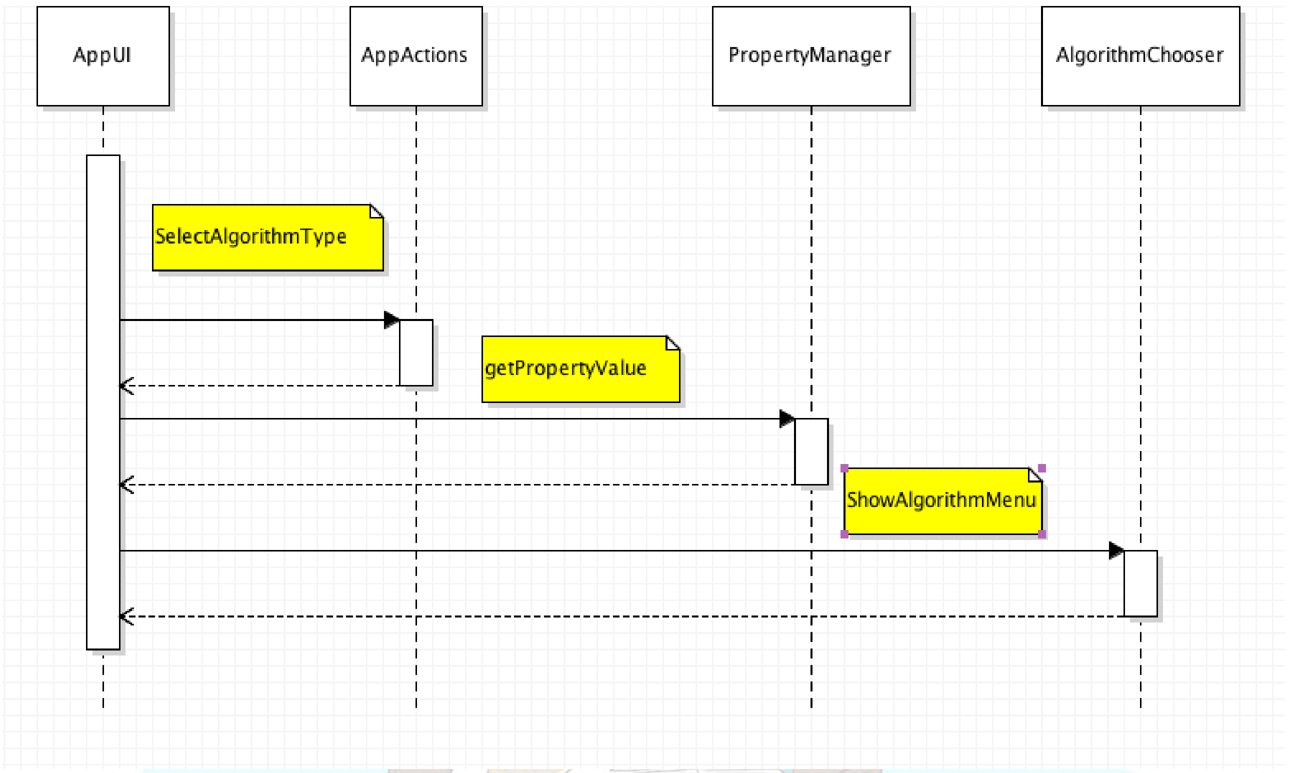


Figure 4.5 Select Algorithm Type UML Sequence Diagram

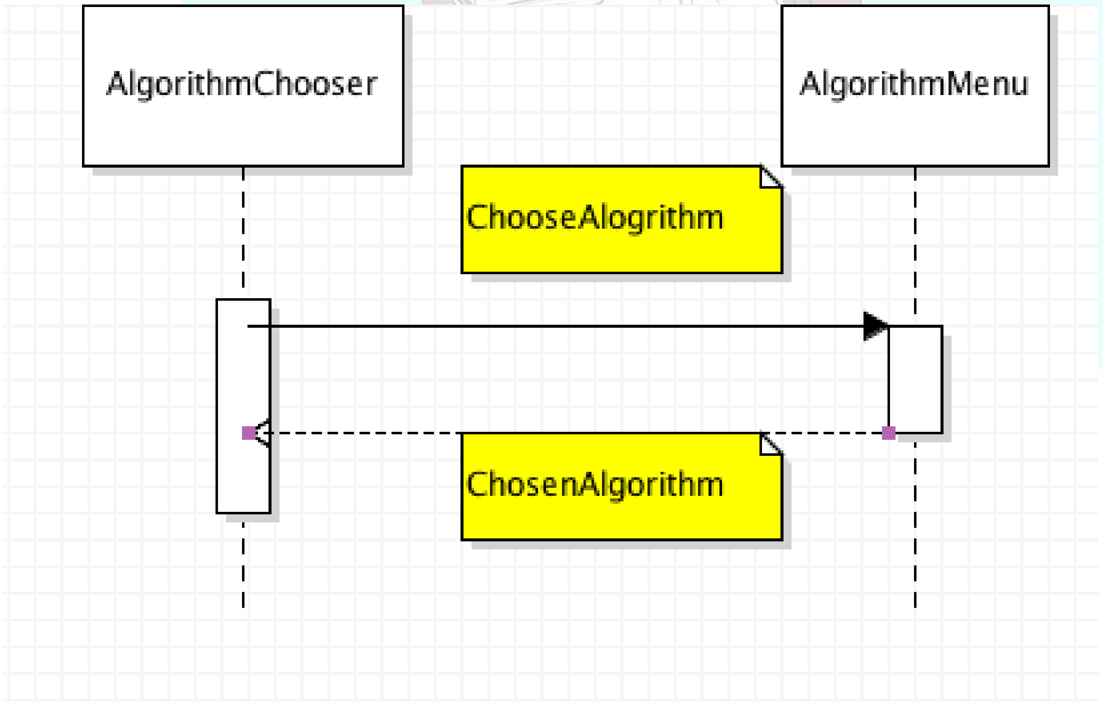


Figure 4.6 Select Algorithm UML Sequence Diagram

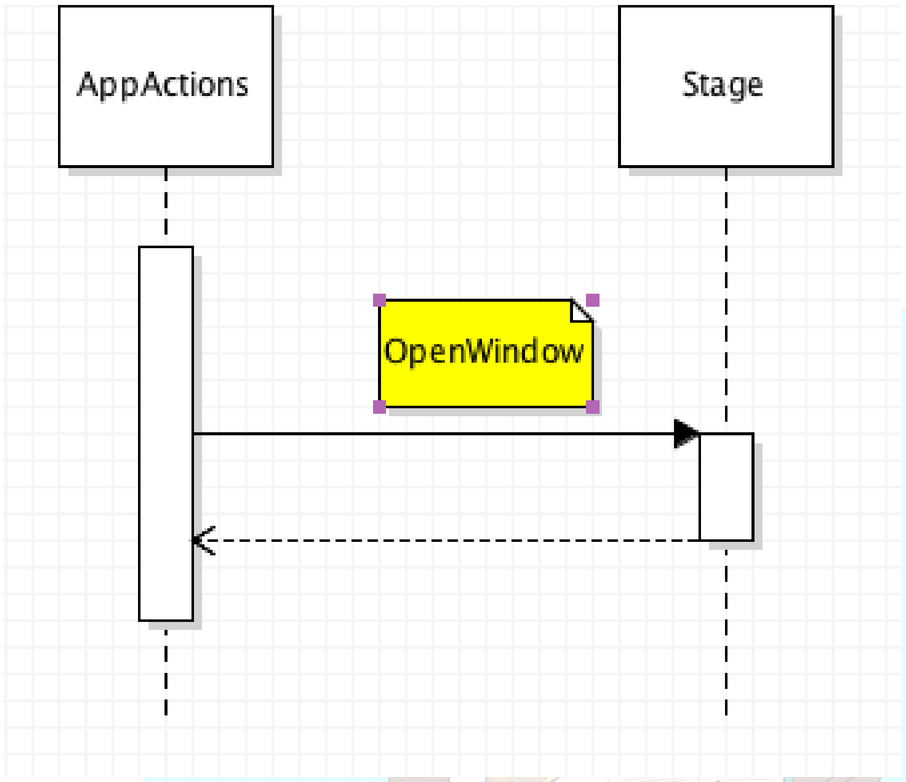


Figure 4.7 Select Algorithm Configuration UML Sequence Diagram

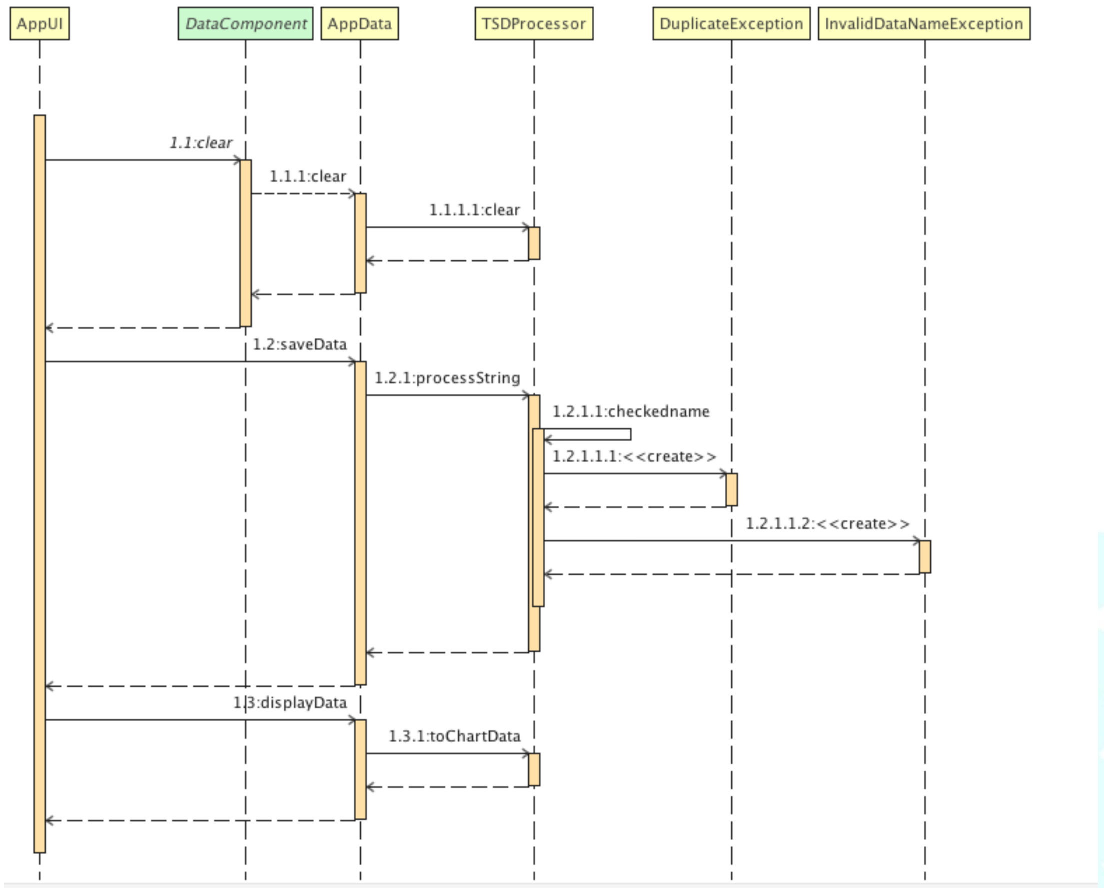


Figure 4.8 Running an Algorithm UML Sequence Diagram

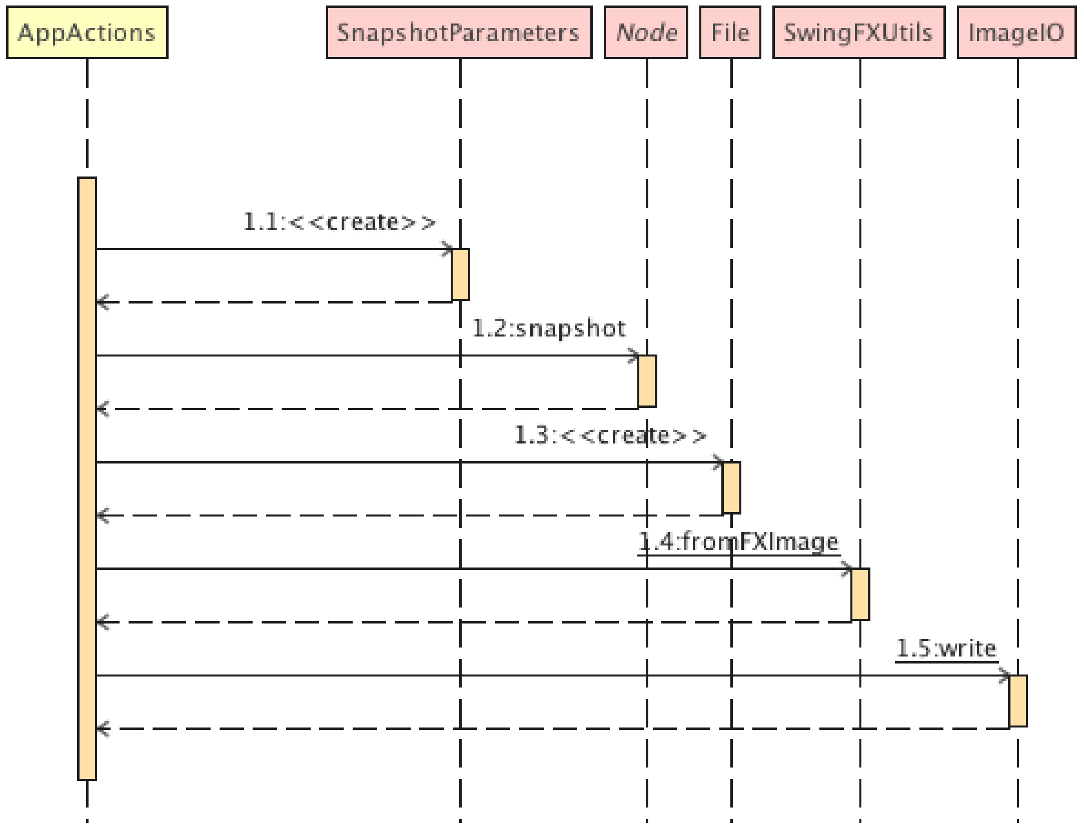


Figure 4.9 Export Data Visualization as Image UML Sequence Diagram

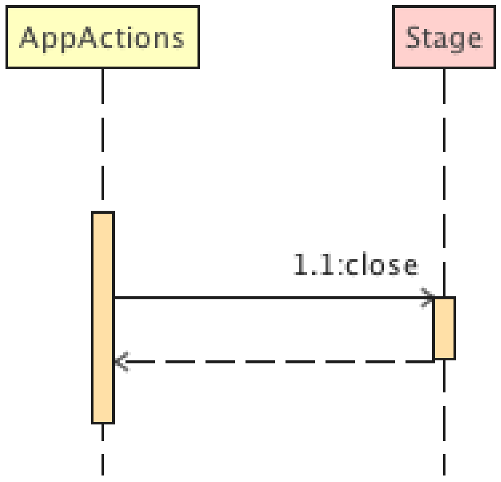
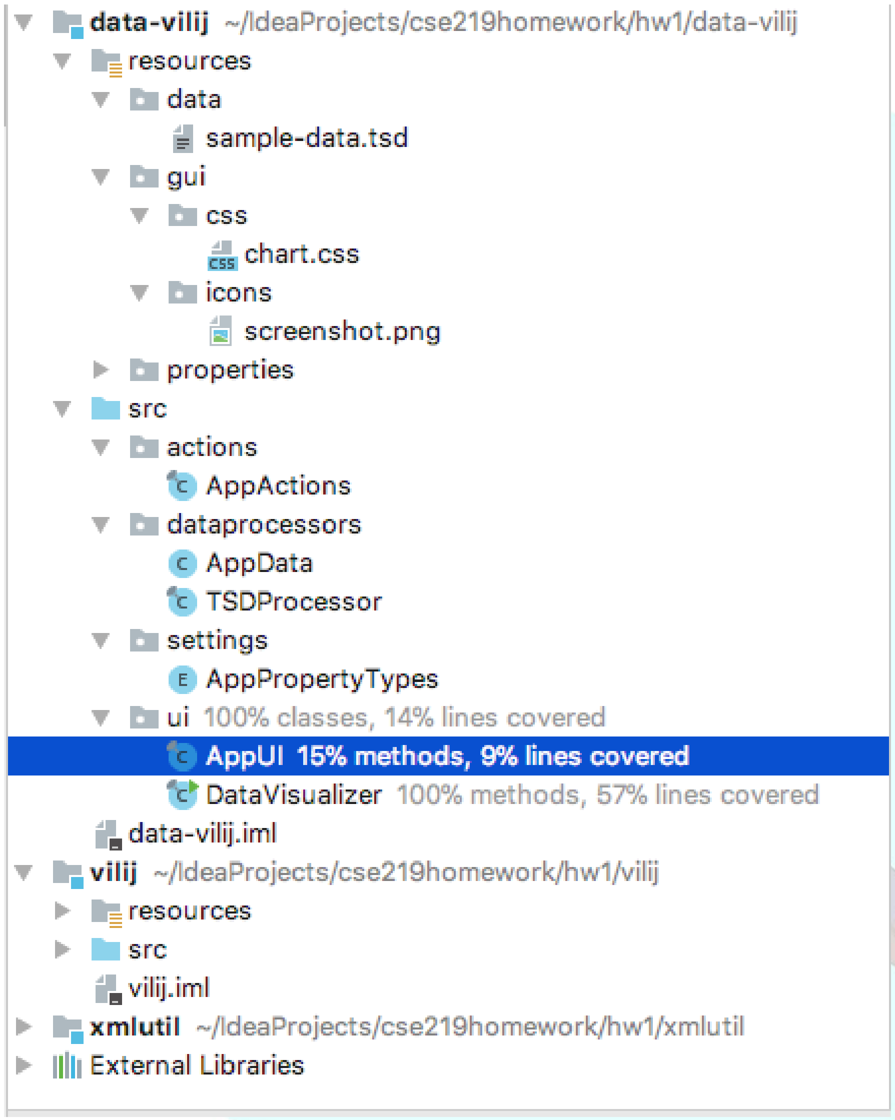


Figure 4.10 Exit Applicaion UML Sequence Diagram

5. File Structure and Formats

We will be using the vilij Framework and xmlutil Framework for this project. Below is the proper file structure for the project. We will use this to design the project.



```
▼ data-vilij  ~/IdeaProjects/cse219homework/hw1/data-vilij
  ▼ resources
    ▼ data
        sample-data.tsd
    ▼ gui
      ▼ css
          chart.css
      ▼ icons
          screenshot.png
    ▶ properties
  ▼ src
    ▼ actions
        AppActions
    ▼ dataprocessors
        AppData
        TSDProcessor
    ▼ settings
        AppPropertyTypes
    ▼ ui  100% classes, 14% lines covered
        AppUI  15% methods, 9% lines covered
        DataVisualizer  100% methods, 57% lines covered
    data-vilij.iml
▼ vilij  ~/IdeaProjects/cse219homework/hw1/vilij
  ▶ resources
  ▶ src
    vilij.iml
▶ xmlutil  ~/IdeaProjects/cse219homework/hw1/xmlutil
▶ External Libraries
```

5.2 app-properties.xml file

This XML file allows us to easily customize the visual aesthetics of the user interface. Note that this properties file is only for the standard high-level controls for an application, so it does not include properties meant for any functionality within the main workspace.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!-- <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."/>
    <property name = "SAVE_FILE_ERROR" value="Save file error"/>
    <!-- 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= "DISPLAY_VALUE" value=" display value"/>
```

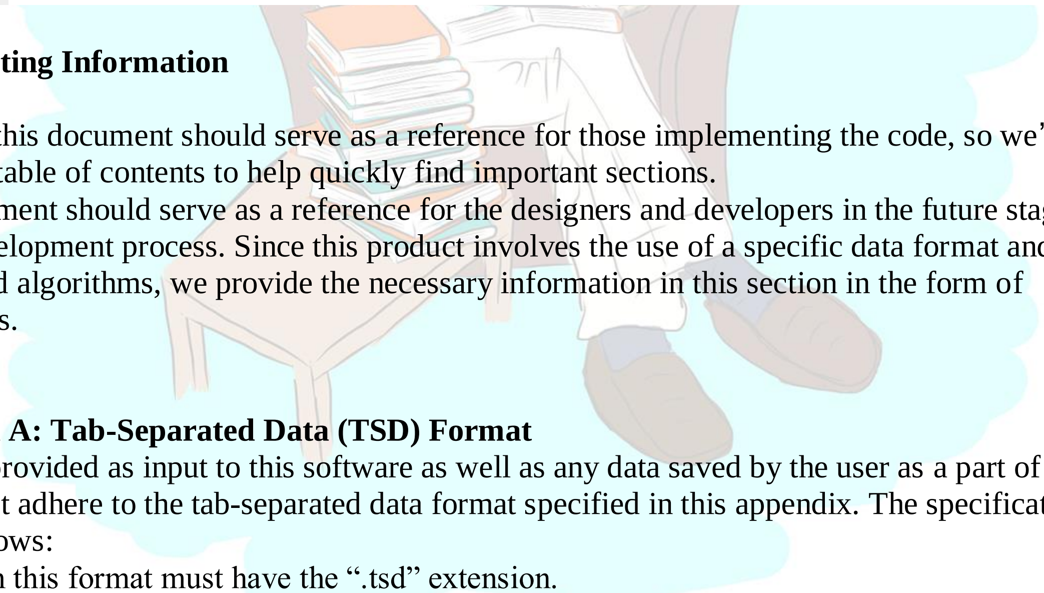
```
        <property name= "DATA_VISUALIZATION" value=" data visualization"/>

        <property name ="SAVE_FILE" value = "Save file"/>
        <property name = "TO_SAVE_THE_FILE_YOU_MUST_CLICK_ON_SAVE" value = "To save the file you must
click on save"/>
        <property name="DATA_FILE" value="Data File"/>

        <property name="GUI_RESOURCE_PATHS" value="gui"/>
        <property name="CSS_RESOURCE_PATHS" value="css"/>
        <property name="CSS_RESOURCE_FILENAMES" value="chart.css"/>


        <property name="READ_ONLY" value="Read-Only"/>

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



6. Supporting Information

Note that this document should serve as a reference for those implementing the code, so we'll provide a table of contents to help quickly find important sections.
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 and specialized algorithms, we provide the necessary information in this section in the form of appendixes.

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.

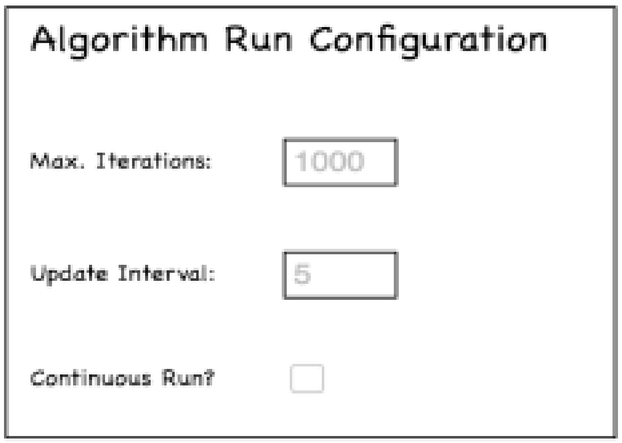
c. Spatial location in the x-y plane as a pair of comma-separated numeric values. The values must be no more specific than 2 decimal places, and there must not be any whitespace between them.
4. There must not be any empty line before the end of file.
5. There must not be any duplicate instance names. It is possible for two separate instances to have the same spatial location, however.

Appendix B: Characteristics of Learning Algorithms

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

B.1 Classification Algorithms

These algorithms ‘classify’ or categorize data into one of two known categories. The categories are the labels provided in the input data. The algorithm learns 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. An overly simplistic example where an algorithm is trying to separate two genders based on hair length (x-axis) and count (y-axis) is shown in Fig. 6.2 (picture courtesy dataaspirant.com). 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, Fig. 6.2. Output of a classification algorithm
• the actual (x, y) position of any instance in its input.
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. These are shown in Fig. 6.1.


Algorithm Run Configuration

Max. Iterations: 1000

Update Interval: 5

Continuous Run? ☐

Figure 6.1

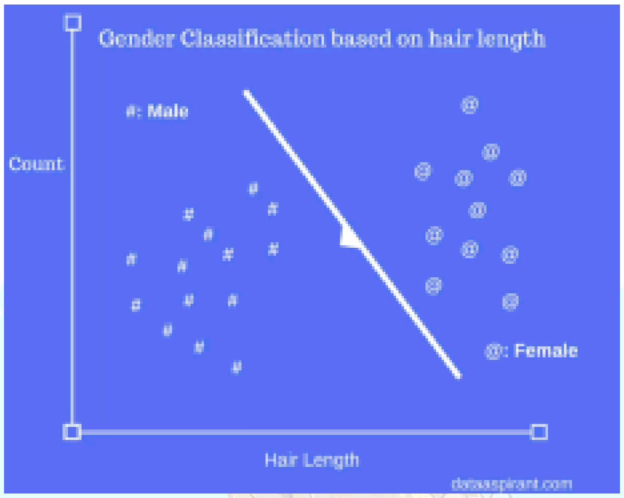

Gender Classification based on hair length
#: Male
@: Female
Count
Hair Length
dataaspirant.com

Figure 6.2

B.2 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. 6.3 and 6.4 below.

Fig. 6.3. 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 6.3

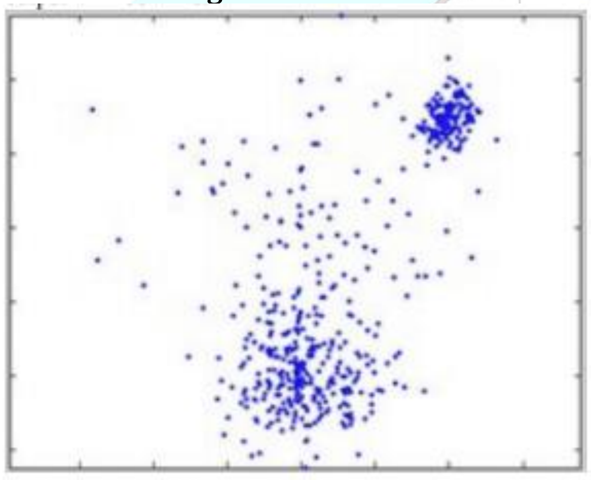


Figure 6.4

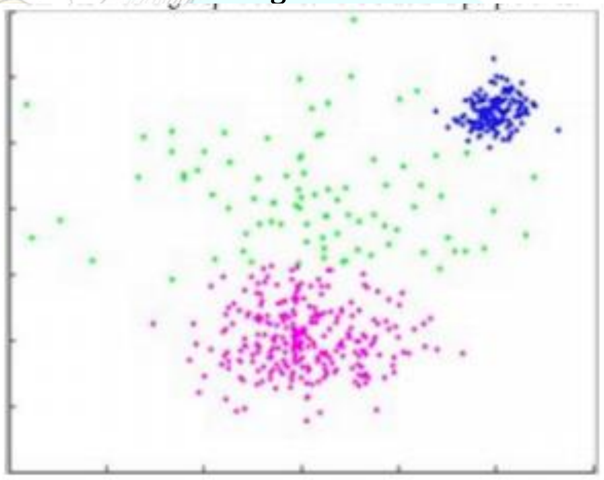


Fig. 6.4. If the number of labels provided to the algorithm as part of its run configuration is 3, then this is what its output might look like. If the run configuration specified 2 labels, the green instances would get split between the blue and purple ones.

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