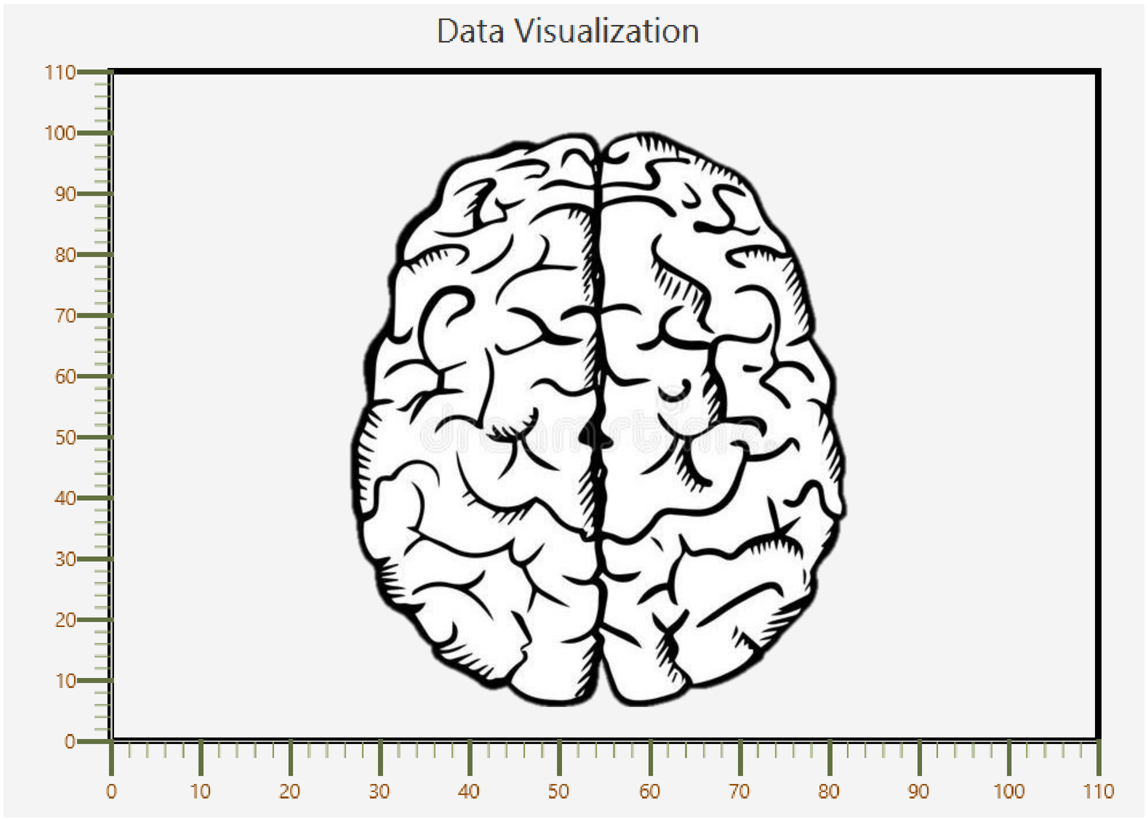
Data Visualization Application

Software Design Description



Author: David Graff
March, 2018
Version 1.0

Abstract: This document describers the software design for the Data Visualization Application for the 3rd to 5th homework and final project of CSE 219. This program is designed to visualize common algorithms associated with AI.

Table of Contents

1 Introduction

This is the Software Design Description for the 3rd to 5th homework and final project of CSE 219. This document format will mirror that of the provided example SDD, which is based on the IEEE Standard 1016-2009 recommendation for software design.

1.1 Purpose

This document lays out the design of the Data Visualization Application. This design uses UML class, sequence, and use case diagrams to give a detailed explanation as to how this project will be laid out. This includes all classes, instance variables, class variables, method signatures needed to build the application, as well as the interaction of objects. The intended audience for this document is the grading TA for CSE 219 and perhaps the instructor as well.

1.2 Scope

This Data Visualization Application is designed to allow a user to easily understand how certain algorithms used in AI manipulate data. At the initialization of this project, this includes only classification and clustering algorithms, but this may expand later. This project will also save and load TSD and SER files from the local host to complete its task. Java is the target language.

1.3 Definitions, acronyms, and abbrev.

GUI – Graphical User Interface, visual controls that the user directly interacts with

Java – A high level programming language

1.4 References

DataVilij^{TM} SRS – The Software Requirements Specification for this project as written by Professor Ritwik Banerjee of Professaur Inc.^{TM}

1.5 Overview

This software Design Description document provides a working design for the Data Visualization Application as described in the SRS. Section 2 will provide the Package-Level Design Viewpoint, specifying the packages and frameworks to be designed. Section 3 will provide the Class-Level Viewpoint, using UML Class Diagrams to show how the classes should be constructed. Section 4 will provide the Method-Level Design Viewpoint, describing how methods will interact with each other. Section 5 provides deployment information like file structures and formats to use. Section 6 provides Supporting Information, if any is necessary.

2 Package – Level Design Viewpoint

As mentioned, this design will encompass the Data Visualization Application. In building this application, we will be using the Java API, as well as TSD and SER files. Following are descriptions of the components to be built, as well as how the Java API will be used to build them.

2.1 Data Visualization Application Overview

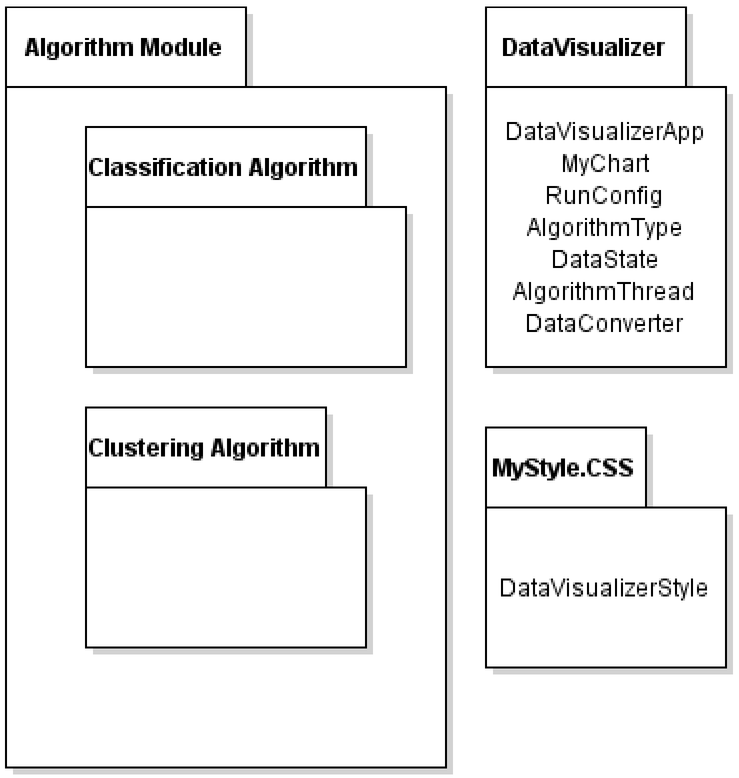


Figure 2.1: Data Visualizer Application Package Overview

2.2 Java API Usage

The Data Visualizer Application will be developed using the Java programming language. This design will make use of the following classes specified in Figure 2.2.

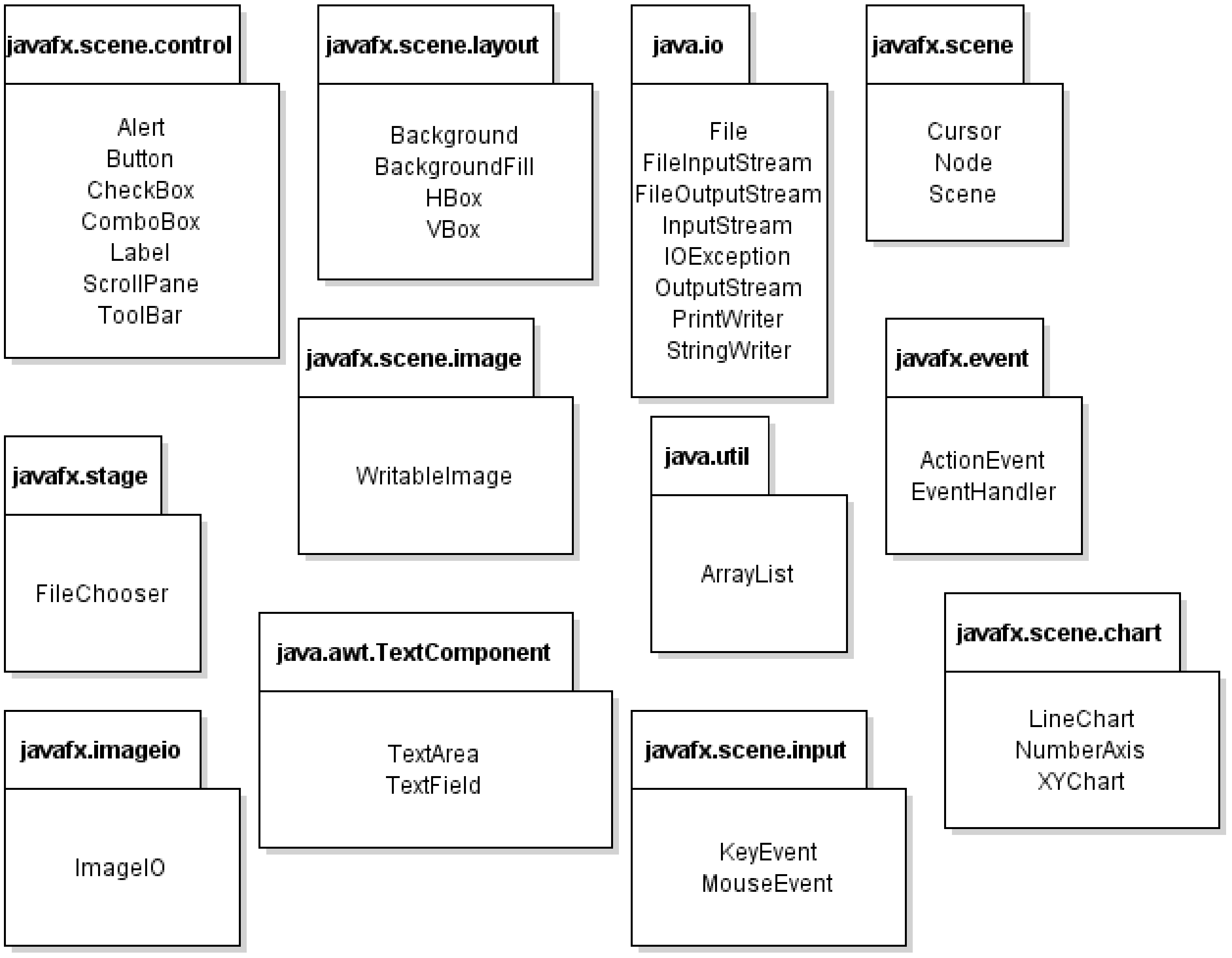


Figure 2.2: Java API Classes and Packages to be Used

2.3 Java API Usage Descriptions

Tables 2.1-2.11 below summarize how each of these classes will be used.

Class/Interface	**Use**
Alert	For informing the user in occurrences of any changes or errors
Button	For selecting different options (e.g. run, save)
CheckBox	To enable and disable TextArea
ComboBox	For selecting an algorithm type
Label	For identifying buttons and the ComboBox's function to the user
ScrollPane	For use if any pane does not fit on the screen
ToolBar	To house the main function buttons(e.g. save, exit)

Table 2.1: Uses for Java API's javafx.scene.control

Class/Interface	**Use**
Background	For setting the background of the pane
BackgroundFill	For filling the background with a color or image
Hbox	For the layout of certain panes going left to right
VBox	For the layout of certain panes going top to bottom

Table 2.2: Uses for Java API's javafx.scene.layout

Class/Interface	**Uses**
File	For finding the path to an external file
FileInputStream	For loading SER and TSD files
FileOutputStream	For saving SER and TSD files
InputStream	For loading data from a SER or TSD file
IOException	For catching errors from InputStream and OutputStream. In case saving or loading is interrupted
OutputStream	For saving data to a SER or TSD file
PrintWriter	For converting an object to a text-output stream
StringWriter	For writing information in a string format

Table 2.3: Uses for Java API's java.io

Class/Interface	**Use**
KeyEvent	For getting information about a key event (e.g. which key was pressed)
MouseEvent	For getting information about a mouse event (e.g. where it was pressed)

Table 2.4: Uses for Java API's javafx.scene.input

Class/Interface	Uses
ActionEvent	For getting information about an action event (e.g. which button was pressed)
EventHandler	For taking action when a certain action occurs

Table 2.5: Uses for Java API's javafx.scene.input

Class/Interface	Uses
Cursor	For setting the action of a mouse
Node	For the purpose of generics
Scene	For control over the applications window

Table 2.6: Uses for Java API's javafx.scene

Class/Interface	Uses
ArrayList	For storing objects, Algorithms in this case

Table 2.7: Uses for Java API's java.util

Class/Interface	Uses
LineChart	To display the data
NumberAxis	To create the chart
XYChart	For declaring new series for the chart

Table 2.8: Uses for Java API's javafx.scene.chart

Class/Interface	Uses
FileChooser	To pick a file to save or load from

Table 2.9: Uses for Java API's javafx.stage

Class/Interface	Uses
ImageIO	To help save the image

Table 2.10: Uses for Java API's javafx.imageio

Class/Interface	Uses
WritableImage	For creating a writable image to save

Table 2.11: Uses for Java API's javafx.scene.image

Class/Interface	Uses
TextArea	To allow the user to edit the data being displayed
TextField	To allow the user to configure algorithms

Table 2.12: Uses for Java API's java.awt.TextComponent

3 Class-Level Design Viewpoint

As aforementioned, this design will encompass the Data Visualizer Application. The following UML Class Diagrams will reflect this.

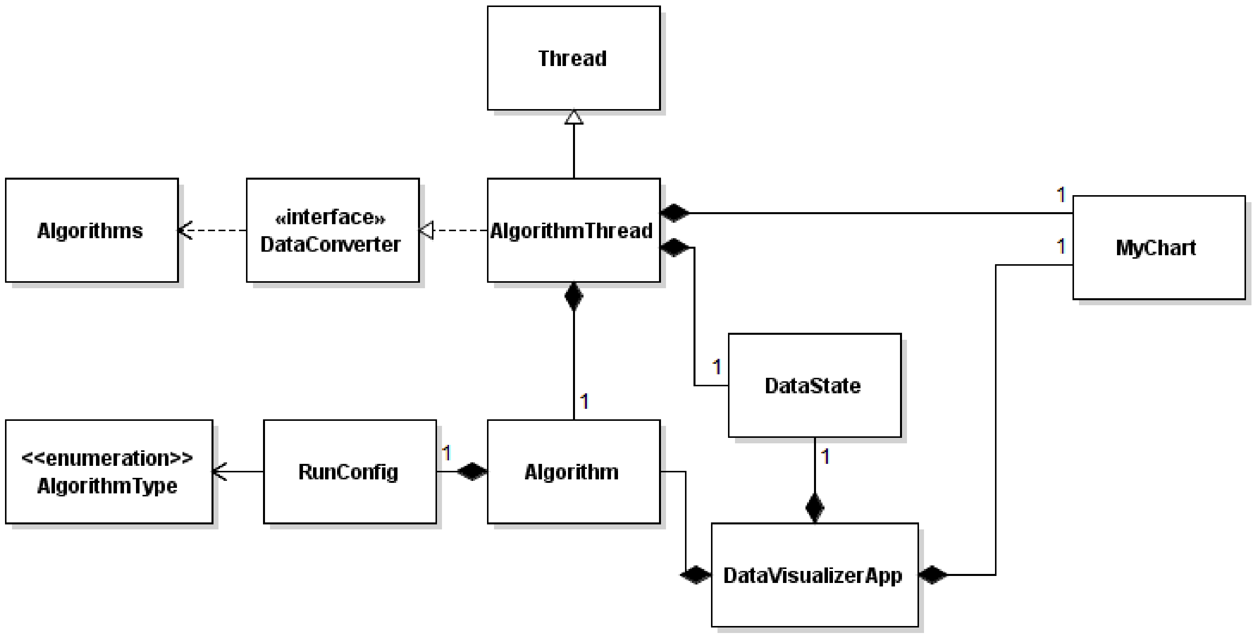


Figure 3.1: Data Visualizer Application UML Class Diagram

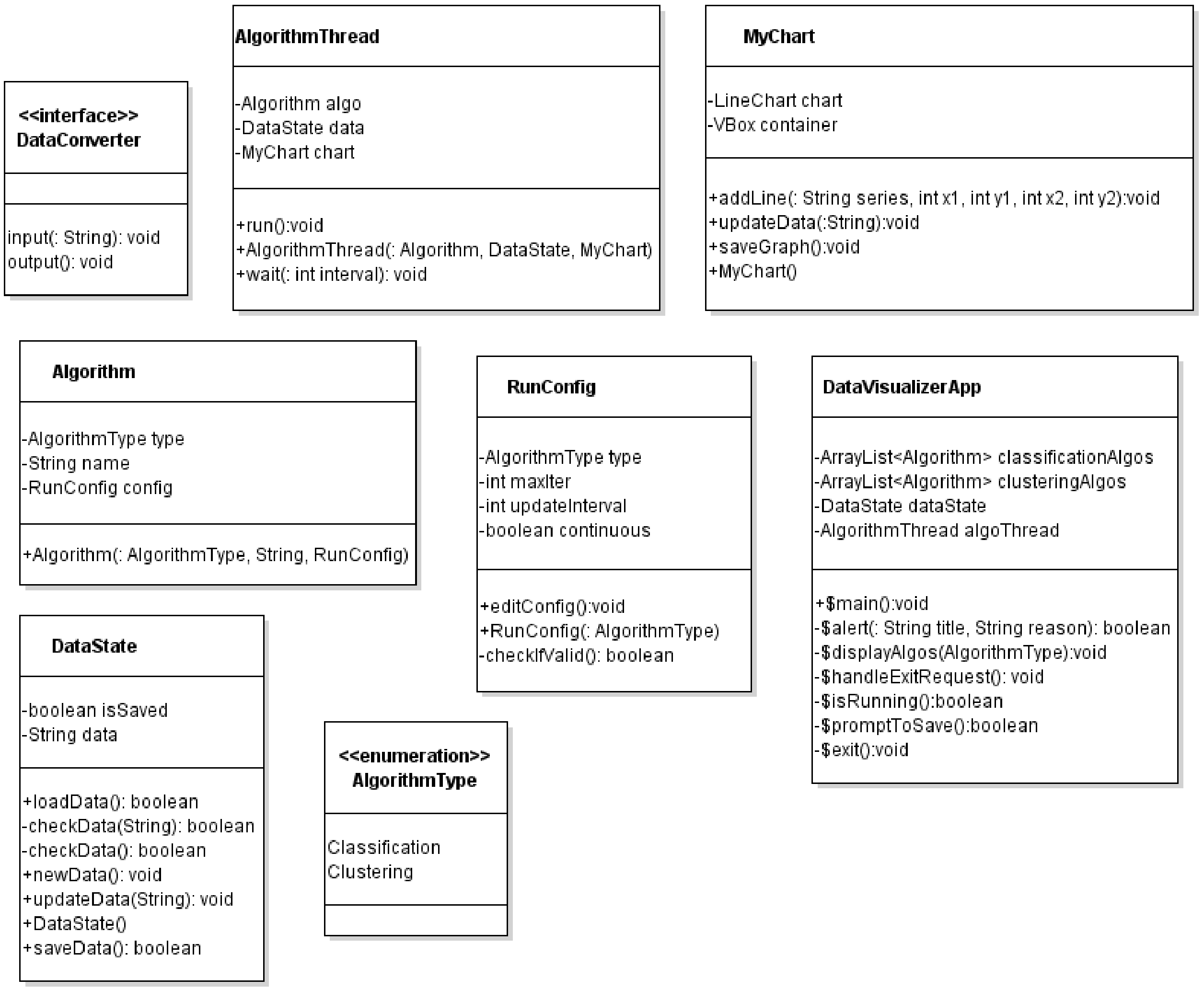


Figure 3.2: Detailed Data Visualizer Application UML Class Diagram

4 Method-Level Design Viewpoint

The following UML Sequence Diagrams describe how data flows through the system in in order to account for the different use cases.

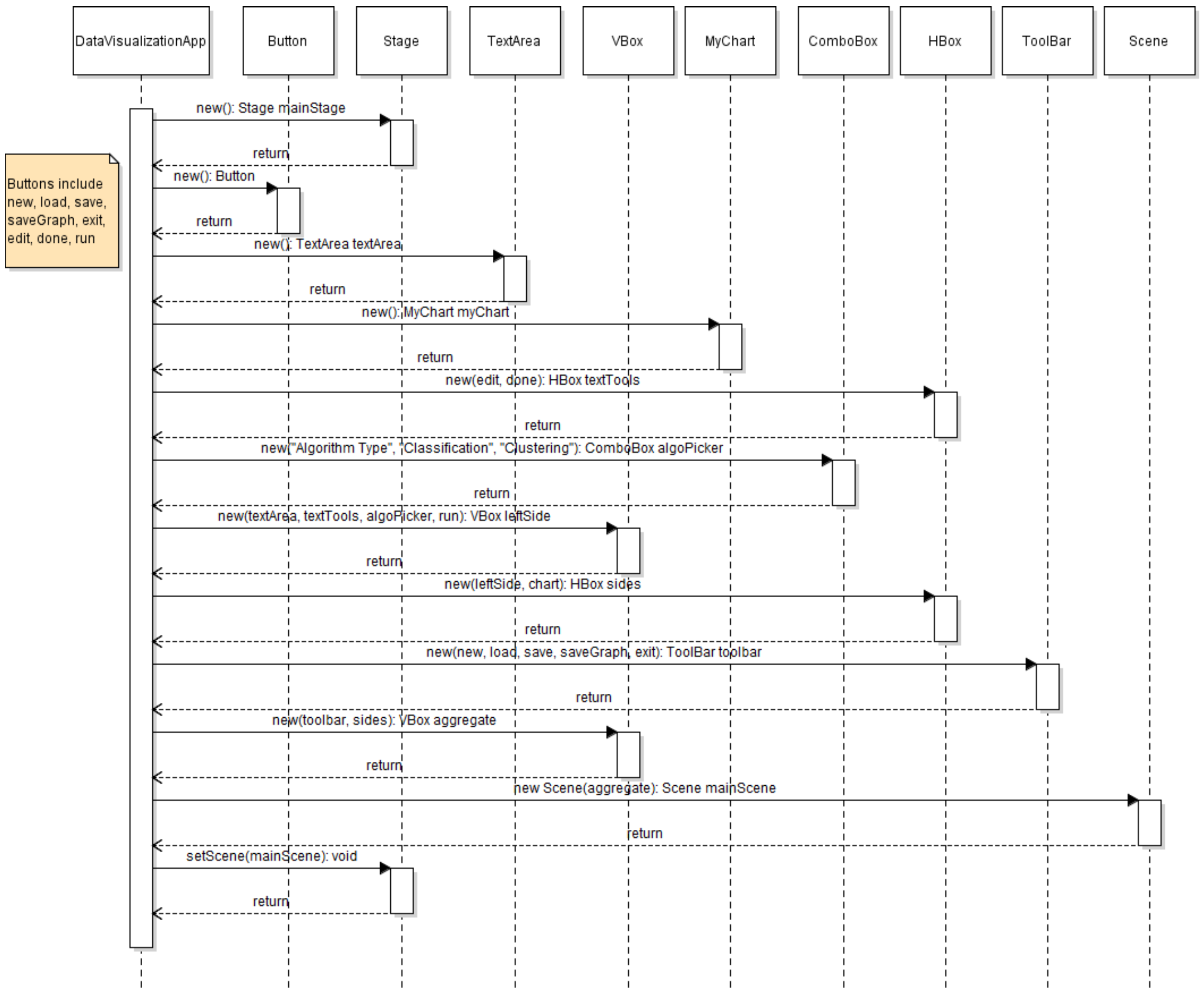

DataVisualizationApp
Button
Stage
TextArea
VBox
MyChart
ComboBox
HBox
ToolBar
Scene
Buttons include new, load, save, saveGraph, exit, edit, done, run
new(): Stage mainStage
return
new(): Button
return
new(): TextArea textArea
return
new(): MyChart myChart
return
new(edit, done): HBox textTools
return
new("Algorithm Type", "Classification", "Clustering"): ComboBox algoPicker
return
new(textArea, textTools, algoPicker, run): VBox leftSide
return
new(leftSide, chart): HBox sides
return
new(new, load, save, saveGraph, exit): ToolBar toolbar
return
new(toolbar, sides): VBox aggregate
return
new Scene(aggregate): Scene mainScene
return
setScene(mainScene): void
return

Figure 4.1: Use Case 1 – Start Application Use

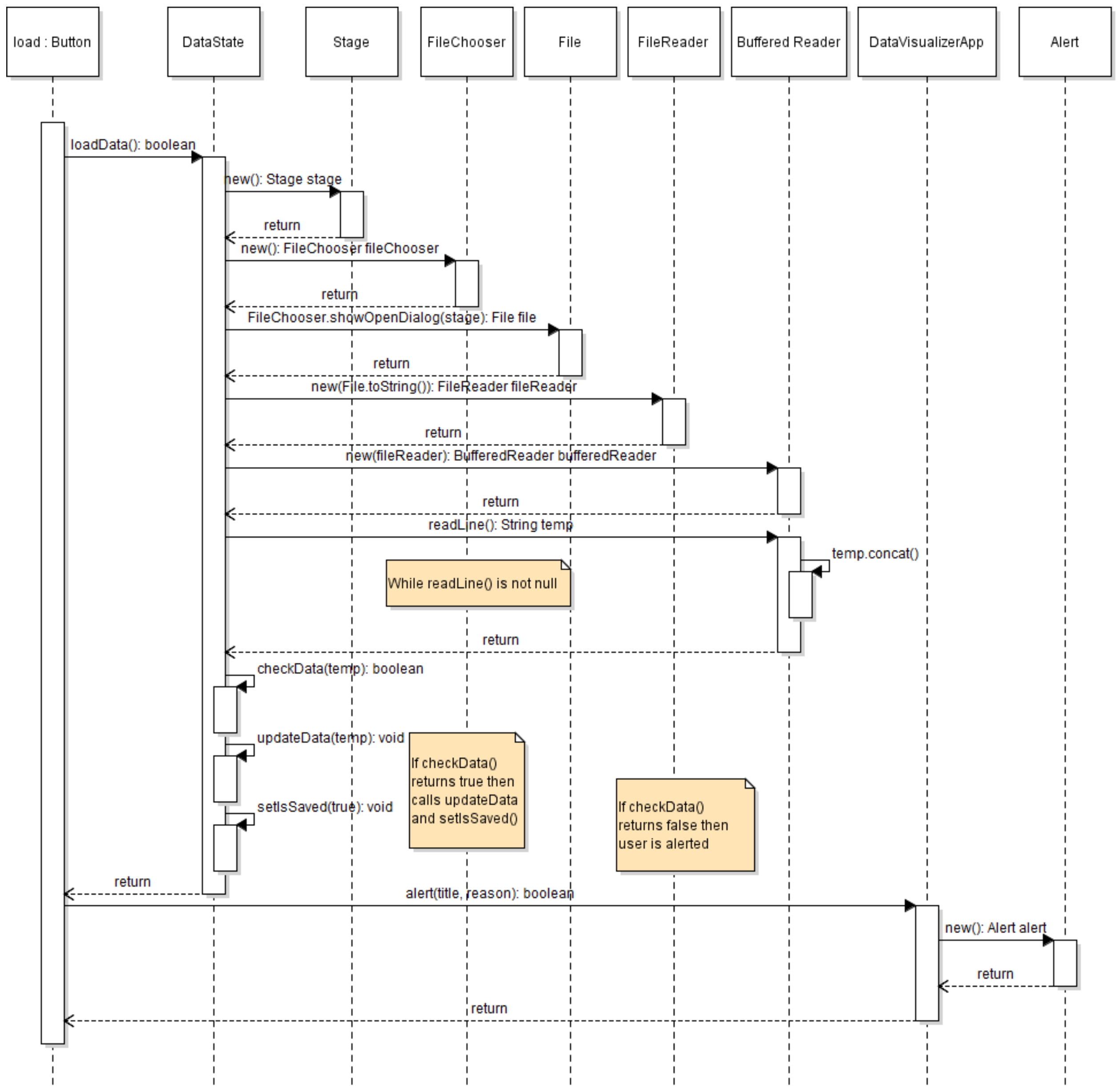

load : Button
DataState
Stage
FileChooser
File
FileReader
Buffered Reader
DataVisualizerApp
Alert
loadData(): boolean
new(): Stage stage
return
new(): FileChooser fileChooser
return
FileChooser.showOpenDialog(stage): File file
return
new(File.toString()): FileReader fileReader
return
new(fileReader): BufferedReader bufferedReader
return
readLine(): String temp
temp.concat()
While readLine() is not null
return
checkData(temp): boolean
updateData(temp): void
If checkData() returns true then calls updateData and setIsSaved()
setIsSaved(true): void
If checkData() returns false then user is alerted
return
alert(title, reason): boolean
new(): Alert alert
return
return

Figure 4.2: Use Case 2 – Load Data

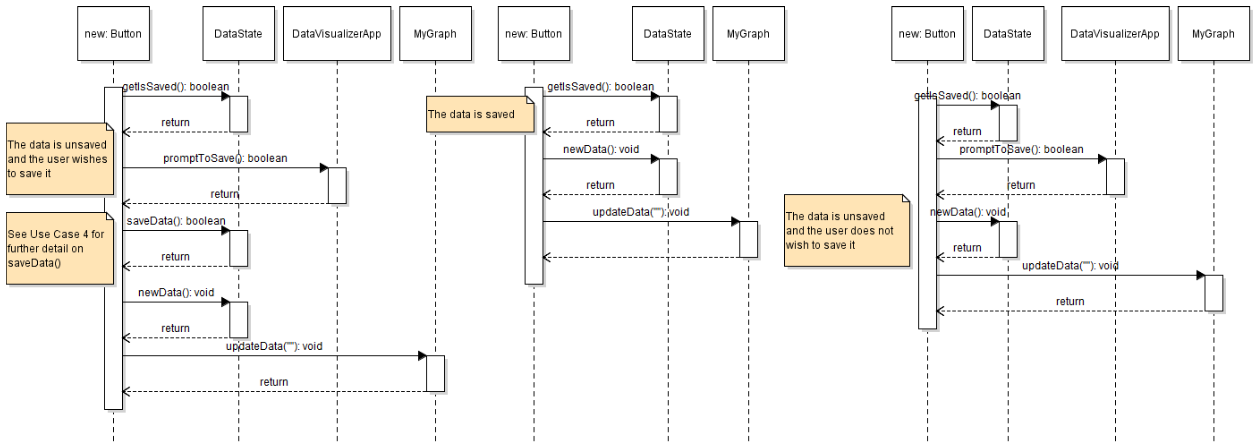


Figure 4.3: Use Case 3 – Create New Data

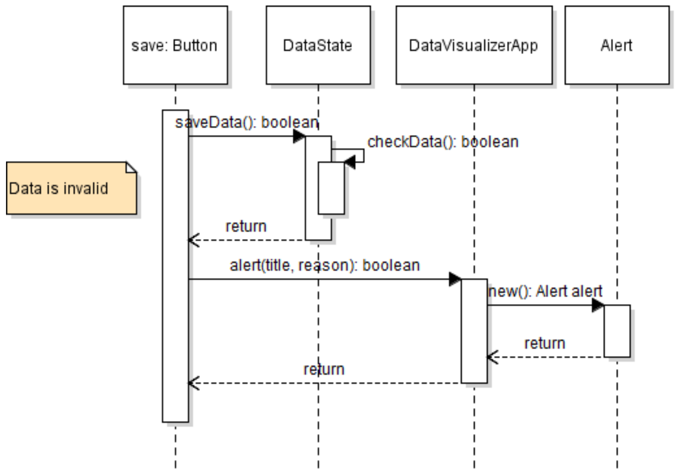


Figure 4.4: Use Case 4 – Save Data, Pt1

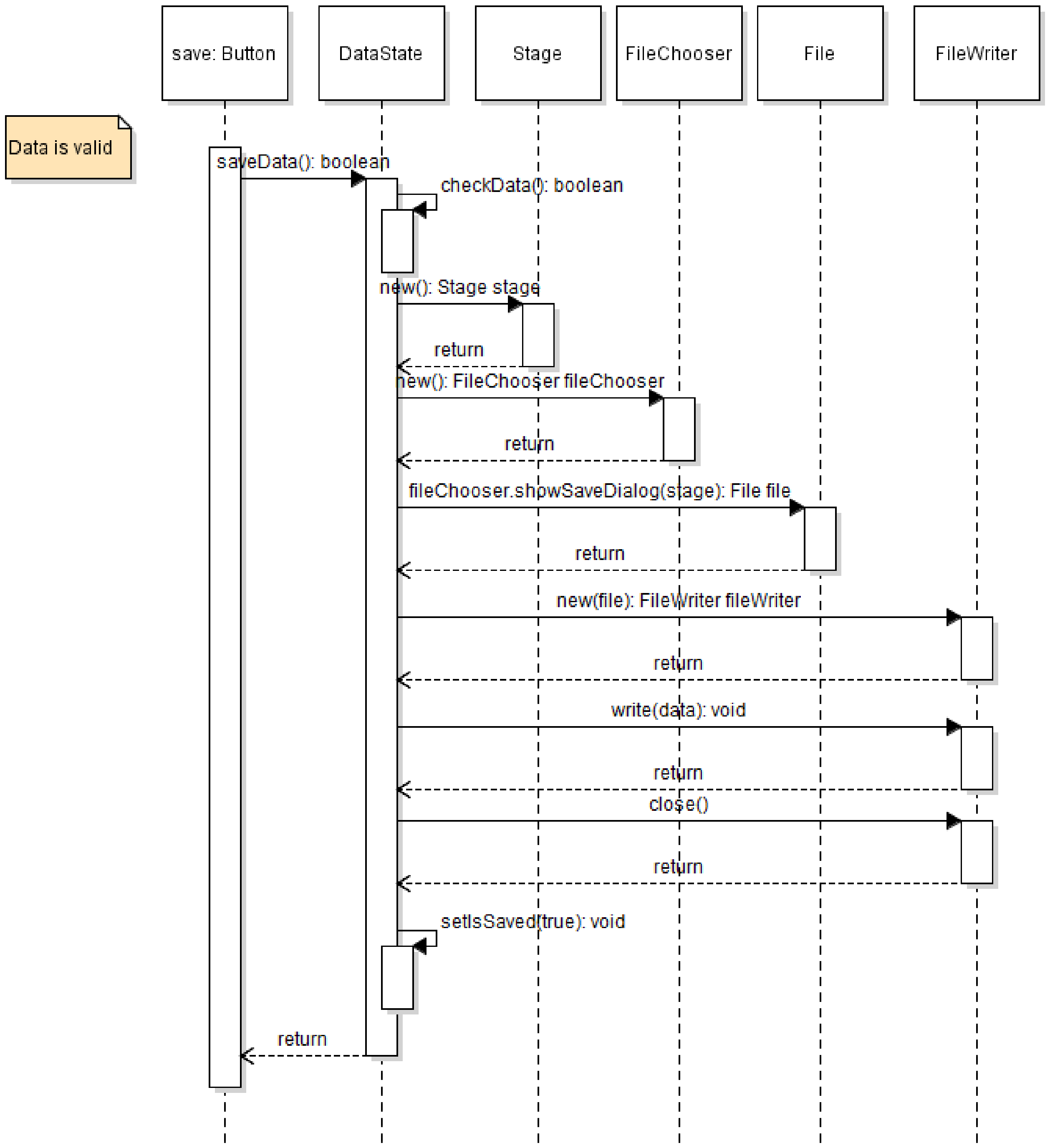


Figure 4.5: Use Case 4 – Save Data, Pt 2

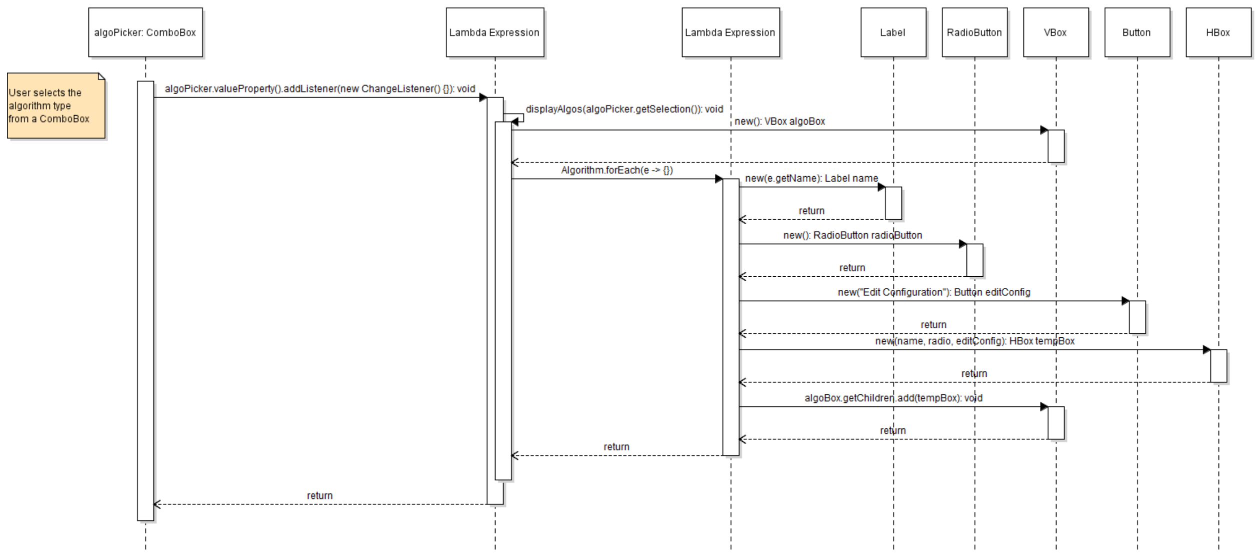

algoPicker: ComboBox
Lambda Expression
Lambda Expression
Label
RadioButton
VBox
Button
HBox
User selects the algorithm type from a ComboBox
algoPicker.valueProperty().addListener(new ChangeListener() {}): void
displayAlgos(algoPicker.getSelection()): void
new(): VBox algoBox
Algorithm.forEach(e -> {})
new(e.getName): Label name
return
new(): RadioButton radioButton
return
new("Edit Configuration"): Button editConfig
return
new(name, radio, editConfig): HBox tempBox
return
algoBox.getChildren.add(tempBox): void
return
return
return

Figure 4.6: Use Case 5 -Select Algorithm Type

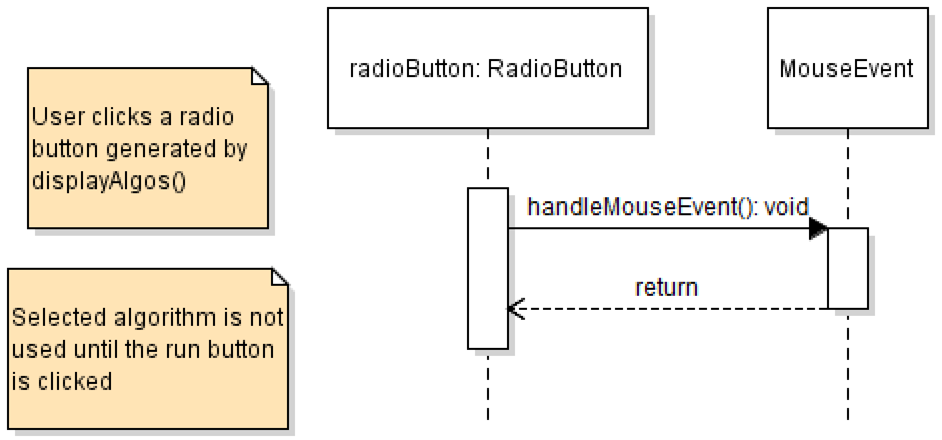

radioButton: RadioButton
MouseEvent
User clicks a radio button generated by displayAlgos()
Selected algorithm is not used until the run button is clicked
handleMouseEvent(): void
return

Figure 4.7: Use Case 6 – Select Algorithm

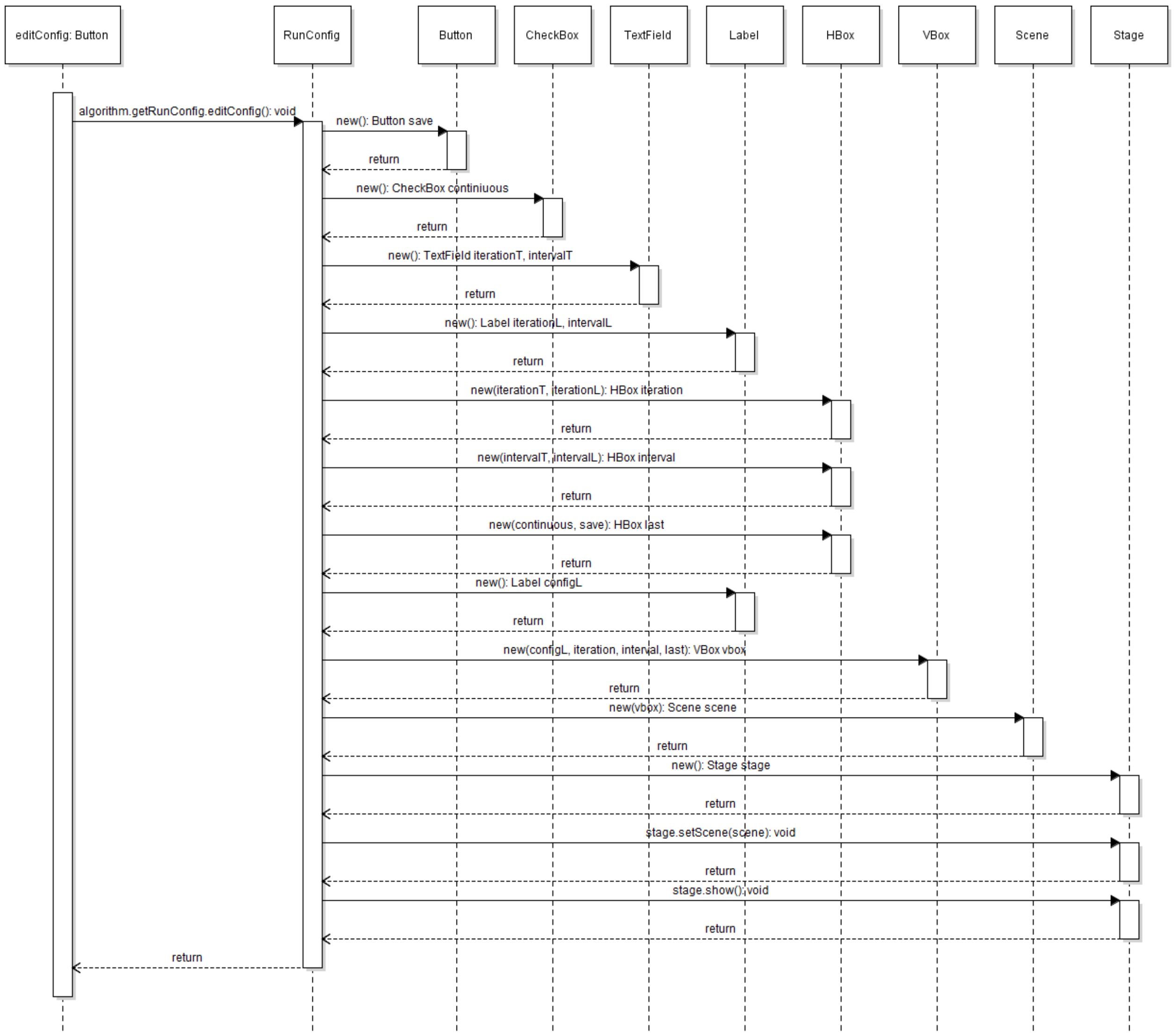

editConfig: Button
RunConfig
Button
CheckBox
TextField
Label
HBox
VBox
Scene
Stage
algorithm.getRunConfig.editConfig(): void
new(): Button save
return
new(): CheckBox continiuous
return
new(): TextField iterationT, intervalT
return
new(): Label iterationL, intervalL
return
new(iterationT, iterationL): HBox iteration
return
new(intervalT, intervalL): HBox interval
return
new(continuous, save): HBox last
return
new(): Label configL
return
new(configL, iteration, interval, last): VBox vbox
return
new(vbox): Scene scene
return
new(): Stage stage
return
stage.setScene(scene): void
return
stage.show(): void
return
return

Figure 4.8 Use Case 7 – Select Algorithm Running Configuration, Pt. 1

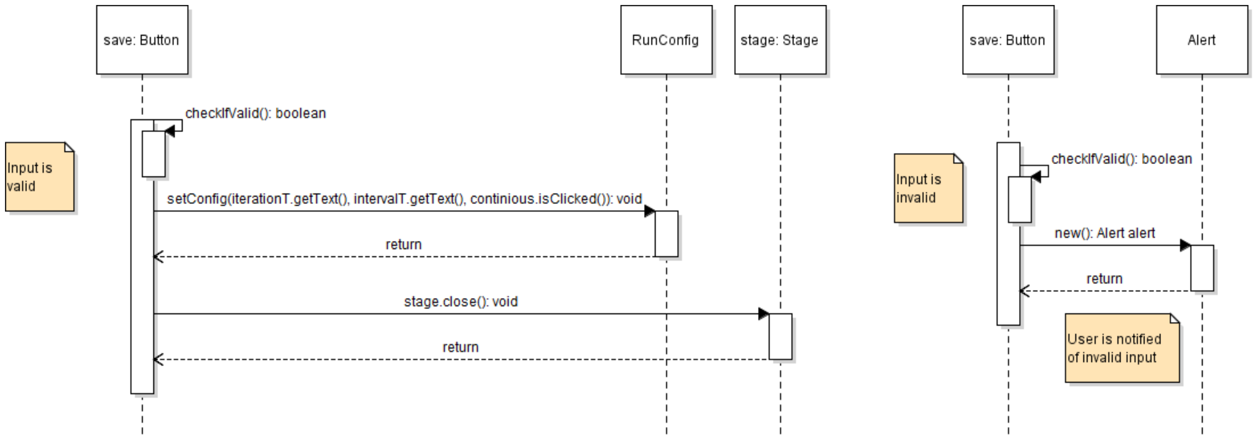


Figure 4.9 Use Case 7 – Select Algorithm Running Configuration, Pt. 2

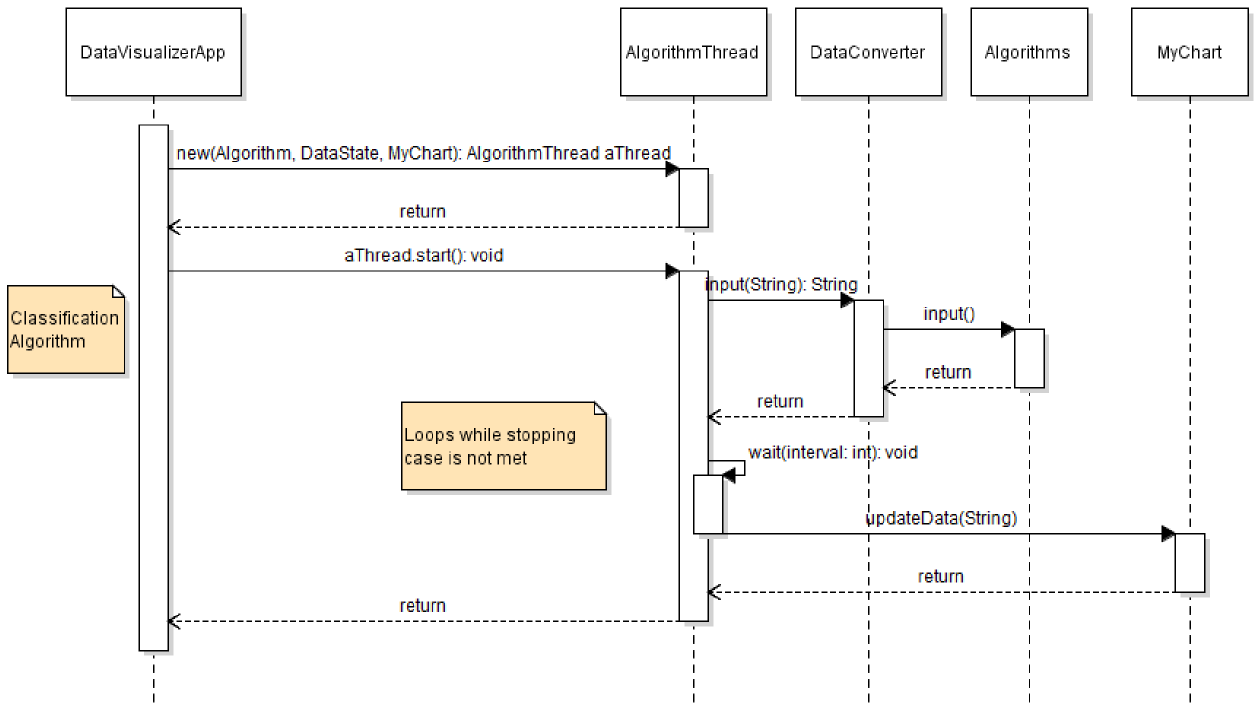


Figure 4.10: Use Case 8 – Running an Algorithm, for Classification Algorithms

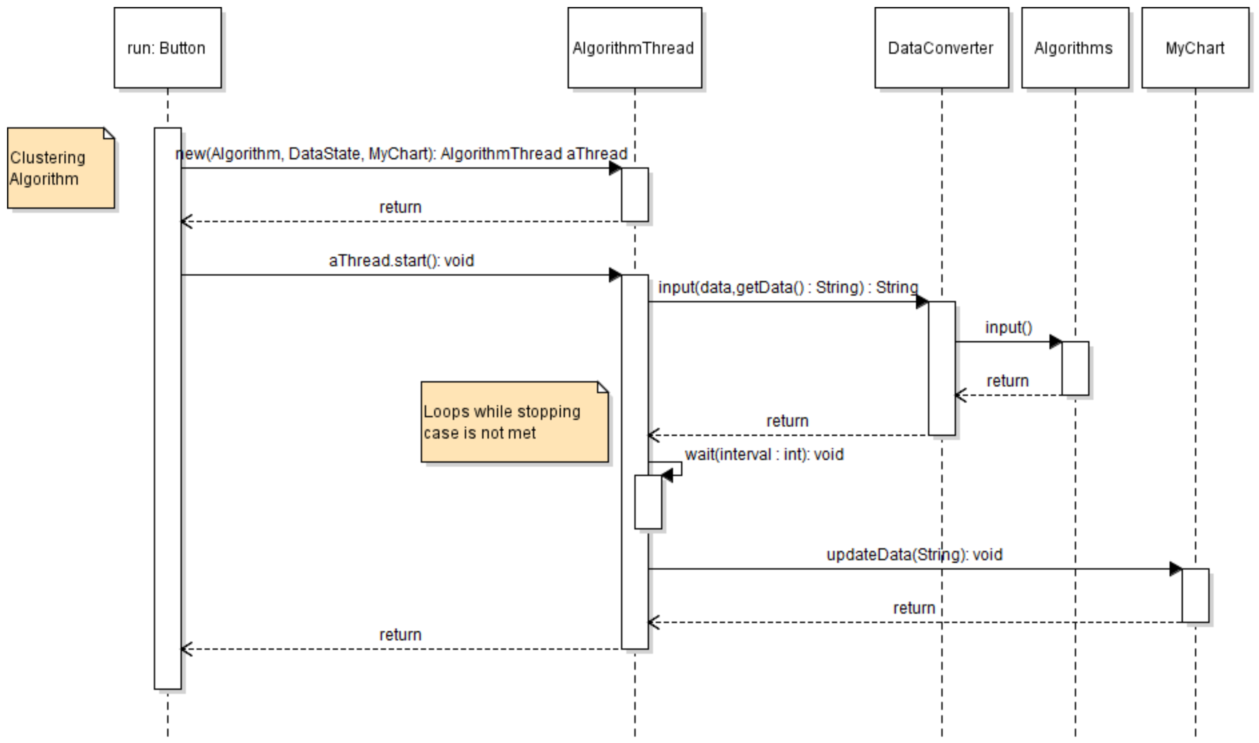

run: Button
AlgorithmThread
DataConverter
Algorithms
MyChart
Clustering Algorithm
new(Algorithm, DataState, MyChart): AlgorithmThread aThread
return
aThread.start(): void
input(data,getData() : String) : String
input()
return
Loops while stopping case is not met
return
wait(interval : int): void
updateData(String): void
return
return

Figure 4.11: Use Case 8 – Running an Algorithm, for Clustering Algorithms

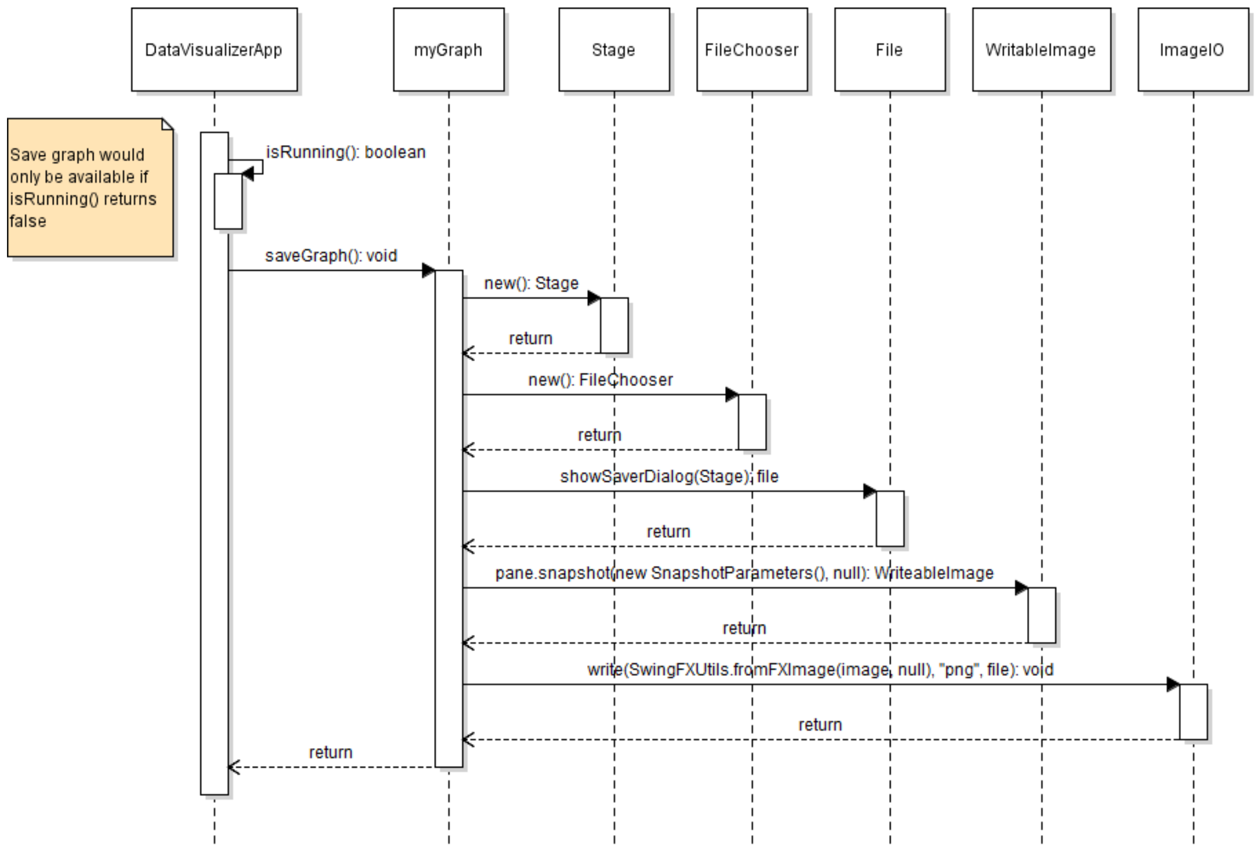

DataVisualizerApp
myGraph
Stage
FileChooser
File
WritableImage
ImageIO
Save graph would only be available if isRunning() returns false
isRunning(): boolean
saveGraph(): void
new(): Stage
return
new(): FileChooser
return
showSaverDialog(Stage): file
return
pane.snapshot(new SnapshotParameters(), null): WriteableImage
return
write(SwingFXUtils.fromFXImage(image, null), "png", file): void
return
return

Figure 4.12: Use Case 9 – Export Data Visualization as an Image

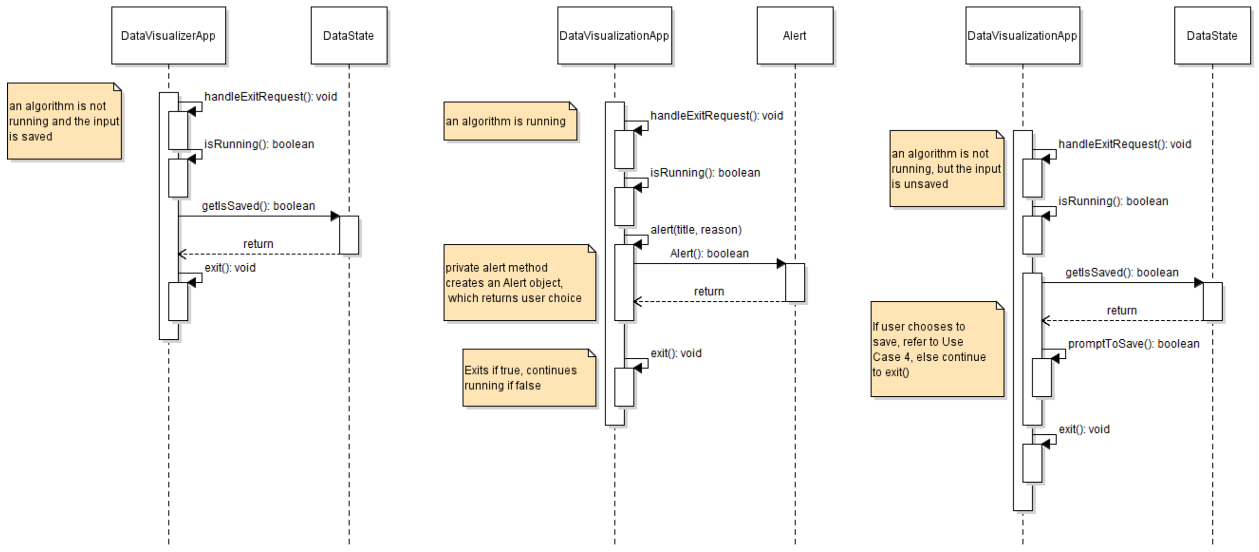

DataVisualizerApp
DataState
an algorithm is not running and the input is saved
handleExitRequest(): void
isRunning(): boolean
getIsSaved(): boolean
return
exit(): void
DataVisualizationApp
Alert
an algorithm is running
handleExitRequest(): void
isRunning(): boolean
alert(title, reason)
Alert(): boolean
return
private alert method creates an Alert object, which returns user choice
exit(): void
Exits if true, continues running if false
DataVisualizationApp
DataState
an algorithm is not running, but the input is unsaved
handleExitRequest(): void
isRunning(): boolean
getIsSaved(): boolean
return
If user chooses to save, refer to Use Case 4, else continue to exit()
promptToSave(): boolean
exit(): void

Figure 4.13 Use Case 10 – Exit Application

5 File Structures and Formats

There are three supporting file types for this application: TSD files by which data can be saved or loaded for processing, SER files in which the algorithms and their configuration files are stored, and a CSS file that maintains the style of the GUI. All three should be found within the application folder. In version 1.0, there is no accompanying art for this program.

6 Supporting Information

There is no relevant supporting information.