WineTracker

Version 2

SFWRENG 2XB3

Software Engineering Practice and Experience: Binding Theory to Practice

Department of Computing and Software

McMaster University

Group 22

Lucentini, Andrew

Zhou, Dios

DiBussolo, Chris

April 10, 2018

Revision History

Revision History		
Date (MM/DD/YYYY)	**Version Name**	**Comments**
04/05/2018	Group22_DesignSpecifications_v0.pdf	- Creation of design specification document
04/07/2018	Group22_DesignSpecifications_v1.pdf	- Rough draft for design specification document. Missing UML state diagrams and small information
04/10/2018	Group22_DesignSpecifications_v2.pdf	- Final draft for design specification document

Team Members		
Name	**Student Number**	**Roles & Responsibilities**
Andrew Lucentini	001430150	• Log admin • Programmer • Researcher • Tester
Chris DiBussolo	400070368	• Project Leader • Programmer • Designer
Dios Zhou	400082351	• Researcher • Tester • Debugger • Analyst

By virtue of submitting this document we electronically sign and date that the work being submitted by all the individuals in the group is their exclusive work as a group and we consent to make available the application developed through SE-2XB3 project, the reports, presentations, and assignments (not including my name and student number) for future teaching purposes.

Contribution

Name	Roles	Contribution	Comments
Andrew Lucentini	• Log admin • Programmer • Researcher • Tester	• Created WineT module • Implemented Search module • Implemented Winelist module • Updated project log	- Contributed to design and requirements specification documents
Chris DiBussolo	• Project Leader • Programmer • Designer	• Implemented DFS module • Implemented Graph module • Implemented Sort module • Implemeted main requirements specification document	- Contributed to design and requirements specification documents
Dios Zhou	• Researcher • Tester • Debugger • Analyst	• Created WineTrackerApp module • Created MouseAdp module • Performed module testing	- Contributed to design and requirements specification documents

NOTE: Any other work was implemented as a group.

Abstract

It is very difficult for wine connoisseurs to discover a wine that suits their desired tastes. One can spend hours looking for a specific wine, only to be left with disappointment as very little information is provided for the wines they are looking for. Not only is there limited information to base a decision off of, but there are no easily accessible reviews written about the wines to aid in decision making. WineTracker solves this issue by recommending a user wines that suit their needs. With over 2 million reviews spanning over 10 years, WineTracker utilizes a database of wine reviews from CellarTracker to create a complex flavour profile for roughly 500,000 wines. Searching through a database of reviews can give a much more accurate representation of a wine's flavour profile as you are getting input from various sources (reviewers) for a single wine. With WineTracker, the user will be provided a list of wines they wish their wine to hold given attributes they input to the program.

The idea behind WineTracker is not just have to do with wine but can be extended to any similar product (such as beer). The idea of building an accurate representation of a product based off many reviews creates a more complex description of the desired product. Thus, WineTracker's searching implementations make it a unique and desirable product.

Contents

1 WineTracker Modules Overview

WineTracker is broken down into six modules. However, to satisfy the assignment specifications there are two extra modules that implement graphing algorithms. The modules included in WineTracker are WineT, Winelist, Sort, Search, WineTrackerApp, and MouseAdp. The reason for splitting WineTracker into multiple modules is to improve the organization and readability of the software. Each module serves a specific purpose, which can be seen in the table below.

WineTracker Modules	
Module	**Purpose**
WineT.java	Used for creating a class that represents a wine. The wine is stored as a name, a type, and a list of reviews. The purpose of this module is to improve WineTracker's modularity.
Winelist.java	Used for loading data from cellarTrackerReviews.txt and storing it in memory. The data is stored as a list of WineT's. The purpose of this module is to bridge the connection between the data and java.
Sort.java	Used for implementing various sorting algorithms utilized throughout WineTracker. The purpose of this module is to allow the same sorting algorithm to be used in multiple locations throughout the program.
Search.java	Used for implementing various searching algorithms utilized throughout WineTracker. The reason for creating this module is to allow the same searching algorithms to be used in multiple locations throughout the program.
WineTrackerApp.java	Used for running a GUI that implements WineTracker. This module contains the main method that runs the program. The GUI is run by simply running the class in eclipse. It is important to note that after running this module, the program may take a minute process the data from cellarTrackerReviews.txt.
MouseAdp.java	Acts as the interface for the mouse event used in the WineTracker GUI. The purpose of this module is to clean up the code in WineTrackerApp.java.
Other Modules	
Graph.java	A module that creates a digraph where the directed edges go to nodes that have a rating between 10-15 points less than the parent node. This module implements features that can be used in future versions of WineTracker
DepthFirstSearch.java	This module implements a dfs search on a digraph of WineTs. This module includes a main, that when run produces a path that would take the user through evenly spaced wines from the starting quality of wine to the worst quality of wine, based on their search. The main method is to simply show how this feature would work with future versions of WineTracker.

1.1 UML class Diagram

Figure 1 shows the UML class diagram showing the static representation of WineTracker classes and relationships between classes. It is important to note that the modules DepthFirstSearch and Graph are not considered as part of the application. All classes use imports that come pre-defined with java. For the simplicity of the diagram, it is assumed that java's List<E>, ArrayList<E>, JPanel and MouseEvent are regular types.

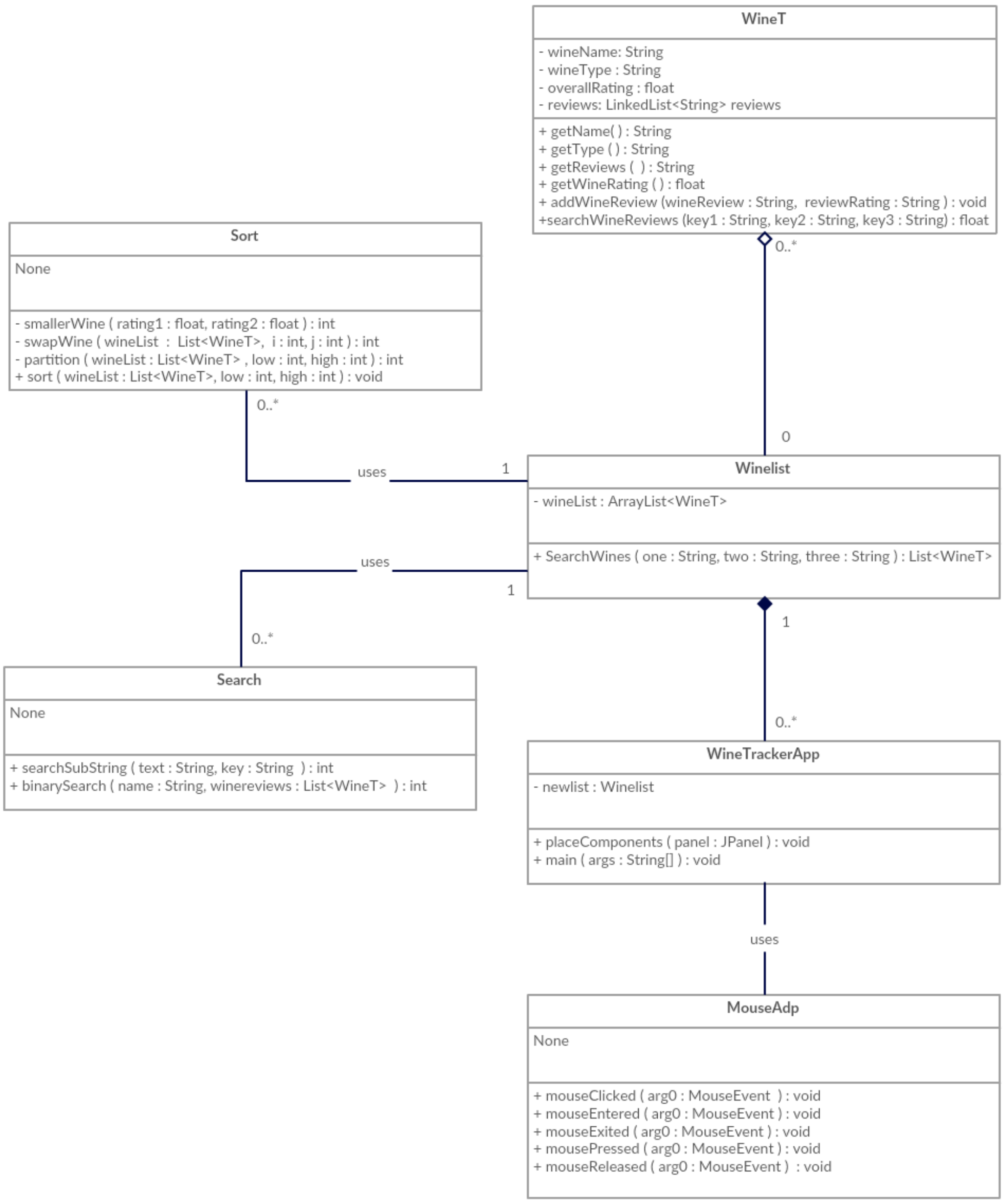
WineT
- wineName: String
- wineType : String
- overallRating : float
- reviews: LinkedList<String> reviews
+ getName( ) : String
+ getType ( ) : String
+ getReviews ( ) : String
+ getWineRating ( ) : float
+ addWineReview (wineReview : String, reviewRating : String) : void
+searchWineReviews (key1 : String, key2 : String, key3 : String) : float
0..*
0
Sort
None
- smallerWine (rating1 : float, rating2 : float) : int
- swapWine (wineList : List<WineT>, i : int, j : int) : int
- partition (wineList : List<WineT> , low : int, high : int) : int
+ sort (wineList : List<WineT>, low : int, high : int) : void
0..*
uses
1
Winelist
- wineList : ArrayList<WineT>
+ SearchWines (one : String, two : String, three : String) : List<WineT>
uses
1
0..*
1
Search
None
+ searchSubString (text : String, key : String) : int
+ binarySearch (name : String, winereviews : List<WineT>) : int
0..*
WineTrackerApp
- newlist : Winelist
+ placeComponents (panel : JPanel) : void
+ main (args : String[]) : void
uses
MouseAdp
None
+ mouseClicked (arg0 : MouseEvent) : void
+ mouseEntered (arg0 : MouseEvent) : void
+ mouseExited (arg0 : MouseEvent) : void
+ mousePressed (arg0 : MouseEvent) : void
+ mouseReleased (arg0 : MouseEvent) : void

Figure 1: UML Class Diagram

2 WineT Module

The WineT module is used to implement an object that represents a wine. The WineT includes a name, a type, a rating, and a list of reviews for that wine.

2.1 WineT Module MIS Specification of Syntax and Semantics

The complete MIS specification for the WineT class is shown below. It is important to note that the type String denotes a sequence of characters and can be written as s = "This is a string", where s[0]=T,s[1]=h, etc.

Wine T Module

Template Module

WineT

Uses

Search, java.util.*

Syntax

Exported Types

WineT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
WineT	String, String	WineT	
getName		String	
getType		String	
getReviews		String	
getWineRating		$\mathbb{R}$	
addWineReview	String, $\mathbb{R}$		
searchWineReviews	String, String, String		
setKey	$\mathbb{N}$		
getKey		$\mathbb{N}$	

Semantics

State Variables

$wineName$: String
$wineType$: String
$overallRating$: $\mathbb{R}$

reviews: seq of String
key : $\mathbb{N}$

State Invariant

None

Assumptions

The constructor WineT is called for each object instance before any other access routine is called for that object. The constructor cannot be called on an existing object. It is also assumed that getKey() can only be called if setKey(i) has been called.

Access Routine Semantics

WineT($name, type$):

- transition: $wineName, wineType, overallRating, reviews := name, type, 0, <>$
- output: $out := self$
- exception: None

getName():

- output: $out := wineName$
- exception: None

getType():

- output: $out := wineType$
- exception: None

getReviews():

- output: $out := ((|reviews| = 0) \Rightarrow (\text{"NO REVIEWS AVAILABLE"}))|(((|reviews| \neq 0) \Rightarrow (reviews[0] + reviews[1]... + reviews[|reviews| - 1]))$ *#In this situation, the + operator is acting as string concatenation. The idea here is to output a string that is the concatenation of all the elements in the reviews sequence as long as the length of the sequence is not 0*
- exception: None

getWineRating():

- output: $out := \frac{overallRating}{|reviews|}$
- exception: None

addWineReview(wineReview, reviewRating):

- transition:
  $reviews := reviews || < wineReview >$
  $overallRating := overallRating + reviewRating$
- exception: None

searchWineReviews(key1, key2, key3):

- output: $out := ((|reviews| = 0) \Rightarrow 0)|((|reviews| \neq 0) \Rightarrow$
  $+(\forall i | i \in [0..|reviews|-1] : (\text{searchSubString}(reviews[i], key1) + \text{searchSubString}(reviews[i], key2) +$
  $\text{searchSubString}(reviews[i], key3)))/|reviews|)$
- exception: None

setKey(i):

- transition: $key := i$
- exception: None

getKey():

- output: $out := key$
- exception: None

2.2 WineT Module and Requirements

The WineT module meets the design requirements of modularity. Storing information about a wine in an ADT creates a layer of abstraction that allows this ADT to be implemented in various areas throughout the application. In addition, the WineT ADT makes the software easier to read and debug.

2.3 WineT UML State Machine Diagram

Figure 2 depicts the UML state machine diagram for the WineT module

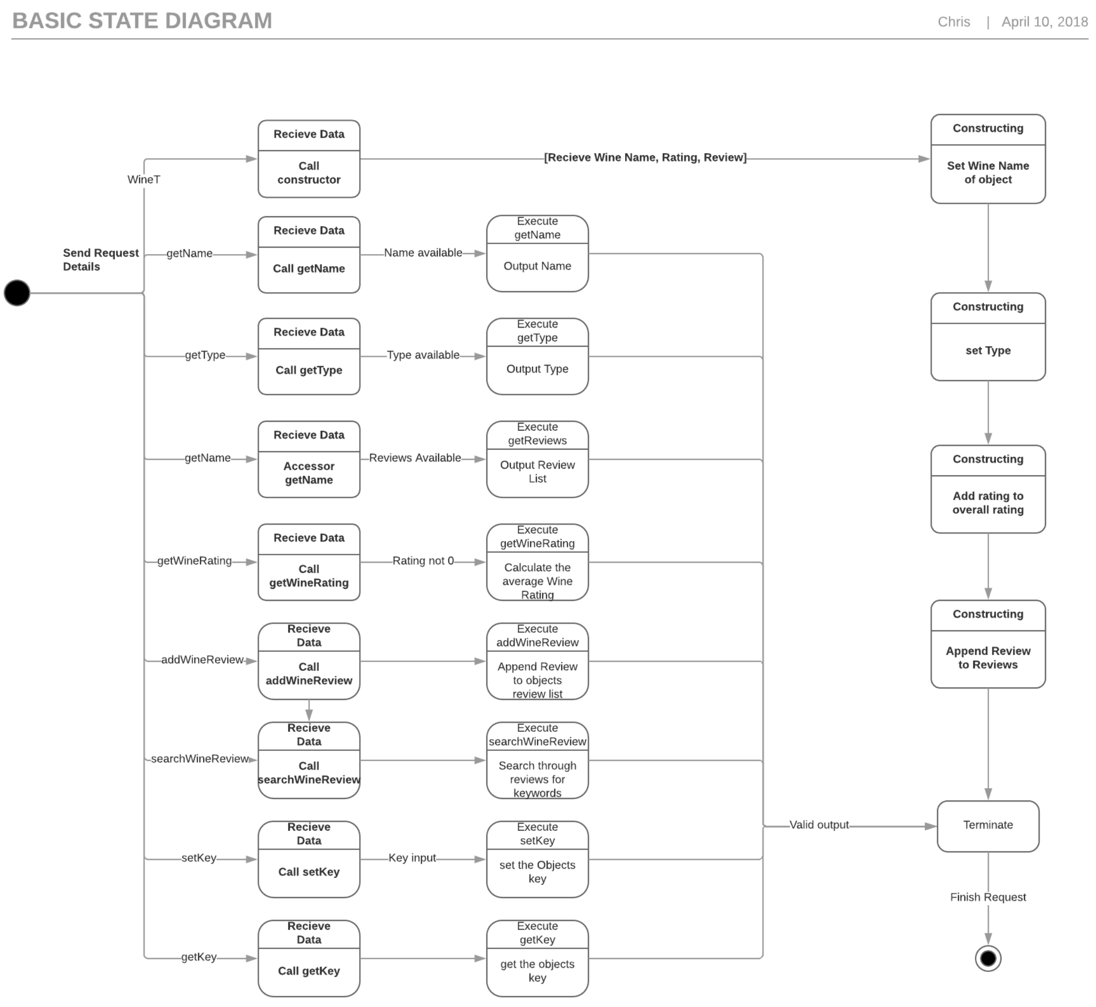

Figure 2: WineT UML State Machine Diagram

3 Winelist Module

The Winelist module is used to extract data from the cellarTrackerReviews.txt and store this data into a List of WineT's.

3.1 Winelist Module MIS Specification of Syntax and Semantics

Winelist Module

Template Module

Winelist

Uses

WineT, java.util.*

Syntax

Exported Types

Winelist = ?

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
Winelist			
SearchWines	String, String, String	String	Set of WineT

Semantics

State Variables

wineList: Seq of String

State Invariant

None

Assumptions

The constructor Winelist is called for each object instance before any other access routine is called for that object. The constructor cannot be called on an existing object. It is assumed that Winelist() is called before SearchWines(one, two, three).

Access Routine Semantics

Winelist():

- transition: $wineList := winesFromTextFile$ *# The winesFromTextFile variable represents a sequence of WineT's that are created by parsing each review in the cellarTrackerReviews.txt file*
- output: $out := self$
- exception: None

SearchWines($one, two, three$):

- output: $out := \{\forall i | i \in [0..|wineList|-1] : (wineList[i].\text{searchWineReviews}(one, two, three) > 2.0) \Rightarrow (wineList[i])\}$ *# The idea is to output a set of WineT's where the set consists of all WineT's from winelist if and only if the result of searchWineReview(one, two, three) for a given WineT is greater than 2.0*
- exception: None

3.2 Winelist Module and Requirements

The Winelist module's primary non-functional concern is efficiency. The majority of the run time was determined through testing to be, and is obviously, the loading of information from the data set into a usable data structure. The Winelist module utilizes binary search when loading information to quickly find the appropriate slot for a wine review to go and store. If something like a linear search were implemented instead, the run time would be far greater than it already is.

4 Sort Module

The sort module provides a series of methods to sort an arrayList (a type defined in java.Util) of WineT objects in a desired order before outputting the final result to the user. The main purpose of the module is to sort the wines in order of highest rating in descending order, to create a "Top 10" list type output to the user.

4.1 Sort Module MIS Specification of Syntax and Semantics

Sort Module

Template Module

Sort

Uses

java.Util.List, WineT

Syntax

Exported Types

N/A

Exported Access Programs

Routine name	In	Out	Exceptions
smallerWine	$\mathbb{R}$, $\mathbb{R}$	$\mathbb{Z}$	
swapWine	Seq of WineT, $\mathbb{Z}$, $\mathbb{Z}$		
partition	Seq of WineT, $\mathbb{Z}$, $\mathbb{Z}$	$\mathbb{Z}$	
sort	Seq of WineT, $\mathbb{Z}$, $\mathbb{Z}$		

Semantics

State Variables

None

State Invariant

None

Assumptions

None

Access Routine Semantics

smallerWine(rating1, rating2):

- output: $out := ((rating1 < rating2 \implies 1)|(rating1 > rating2 \implies -1))$
- exception: none

swapWine($wineList, i, j$):

- transition: $wineList[i], wineList[j] := wineList[j], wineList[i]$
- exception: none

partition($wineList, low, high$):

- output: $out := (j : wineList[0..j-1] < wineList[j] \wedge wineList[j+1..|wineList|-1] > wineList[j])$
- exception: none

sort($wineList, low, high$):

- transition: $wineList := (wineList | i \in [0..|wineList|-1] : wineList[i] > wineList[i+1])$
  (sorts the wineList in descending order)
- exception: None

4.2 Sort Module and Requirements

The sort module meets the design requirements of accuracy and efficiency. Due to the large size of the Cellar Tracker data set, the most efficient in-place algorithm, quicksort, was implemented to meet the need of an efficient method to sort the possibly large amount of wines to outputted. Sorting allows the software to output to the user the best possible wines to choose from based on user reviews, based on what the user was looking for. This ordering of the output increases the accuracy of the results the user is receiving by ensuring that the wines being recommended are indeed good choices for the user.

5 Search Module

The search module is used to maintain the order of the data array as data is loaded from the data set into memory wine objects.

5.1 Search Module MIS Specification of Syntax and Semantics

Search Module

Template Module

Search

Uses

java.Util.List, WineT

Syntax

Exported Types

N/A

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
searchSubString	String, String	$\mathbb{Z}$	
binarySearch	String, List<WineT>		

Semantics

State Variables

None

State Invariant

None

Assumptions

None

Access Routine Semantics

searchSubString($text, key$):

- output: $out := ((text.\text{contains}(key) \implies 1)|(\neg text.\text{contains}(key) \implies 0))$
- exception: none

binarySearch($name, wineReviews$):

- output: $out := (i : wineReviews[i].\text{getName}() = name)|$
  $(i : wineReviews[i-1].\text{getName}() < name \wedge wineReviews[i+1].\text{getName}() > name)$

  (outputs where the wine with that name IS in the array, or where it should be if not already there.)
- exception: none

5.2 Search Module and Requirements

This module contributes to the efficiency requirement of the design. By utilizing binary search to maintain the order of the data, it ensures that the data reviews being read from the Data set will not create duplicate wines in memory. By ensuring that every wine has only one unique object with individual reviews for the same wine being stored in such a unique object, it minimizes the space required for storage and thus the time it takes to perform operations on that data, such as sorting and further searching.

6 Graph and DepthFirstSearch Modules

An in depth breakdown of the Graph and DepthFirstSearch modules is not required because they were not part of the final WineTracker application. The reason these modules were implemented is to provide features for possible future versions of WineTracker. The Graph modules intended purpose is to provide the user with a sort of "Wine Tour" that takes the user down a path of wine's to experience different wine's at each user rating, to experience the range of quality. The graph uses the output array that the user would see after a search as the input to make a graph. The directed edges go to nodes that have a rating between 10-15 points less than the parent node. By performing a dfs from one point to another, the user will receive a path that would take them through evenly spaced wines from the starting quality of wine to the worst quality of wine, based on their search. However, due to time constraints and unsatisfactory results and formatting from the implementation, the module will not be included in the final product of wine tracker and is just extra. A sample demonstration has been prepared in the main function of Graph.java if a demonstration is desired.

7 WineTrackerApp Module

The WineTrackerApp module implements a demo application of WineTracker. This demo acts as a preliminary prototype for the future mobile application that will be developed for WineTracker. This Module contains a main method, that when run will produce the WineTracker GUI. It is important to note that the program may take about a minute to load the information from the text file into java.

7.1 WineTrackerApp UML State Machine Diagram

Figure 3 depicts the UML state machine diagram for the WineTrackerApp.

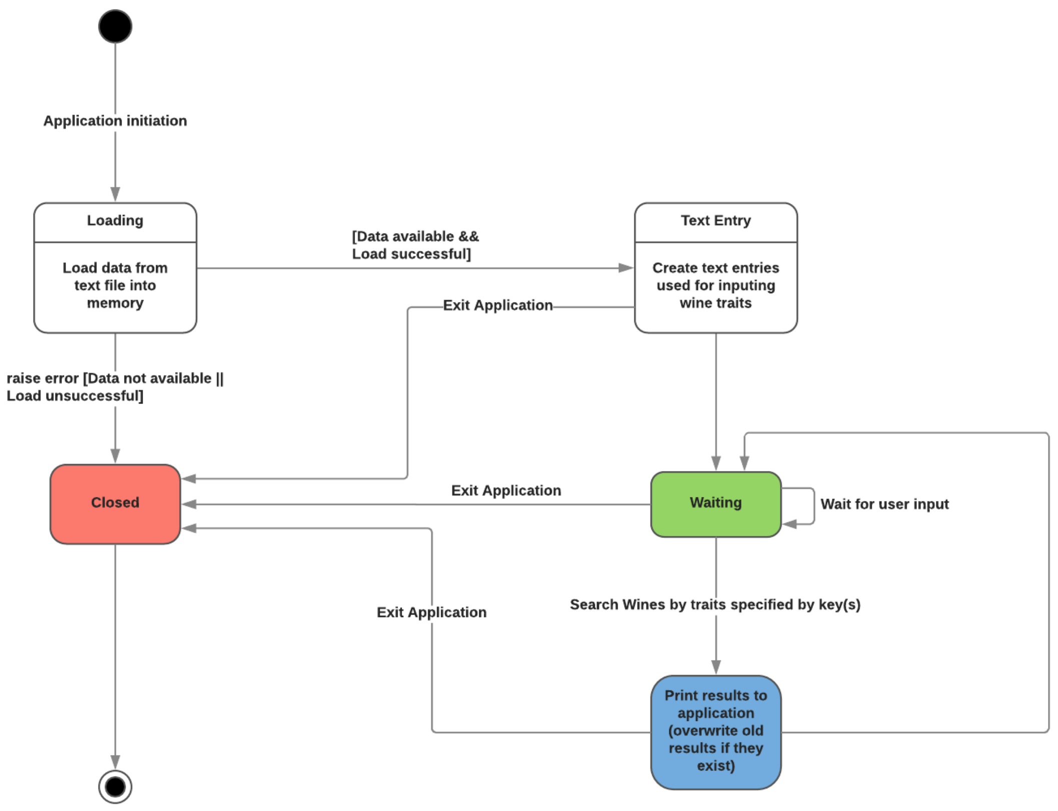
Application initiation
Loading
Load data from text file into memory
[Data available && Load successful]
Text Entry
Create text entries used for inputing wine traits
Exit Application
raise error [Data not available || Load unsuccessful]
Closed
Exit Application
Waiting
Wait for user input
Exit Application
Search Wines by traits specified by key(s)
Print results to application (overwrite old results if they exist)

Figure 3: WineTrackerApp UML State Machine Diagram

8 MouseAdp Module

The MouseAdp module is a very short module that simply implements the interface for a mouse event in the WineTrackerApp GUI. The main use of the module is to simplify the code in WineTrackerApp. The module uses two imports from the java standard library, specifically java.awt.event.MouseEvent and java.awt.event.MouseListener.There are no private entities in the module. Each method in the module specifies how the mouse clicking event should be administered.

9 Review and Evaluation

The design of the WineTracker application utilizes many of the major software engineering principles. Modularity, efficiency, accuracy and reliability were the primary focus of the application. The application shows great promise, as the use of the java application to load the data once and perform many instantaneous searches off of that data makes the application practical for use. However, the inital run time remains impractical as typical applications takes no more than 30 seconds max to open even on a mobile device.

With more time the primary focus of the application would be to introduce a method of cutting down the load time of the application. Possible solutions would be to load the necessary information on installation of the app, so that the application opens and can be used instantaneously upon the user's request. Another design feature that would be included should the application be further developed is the option to change the filters for the sorting of the output. Currently, the user will always see a list of wines sorted by the highest user rating. Traditional search engines however, tend to offer multiple options such as number of search hits, most reviews, relevancy and so on. Lastly a feature that would be implemented is a side bar window that could be opened by the user. The user could then enter rating values they would want to experience and the graphing algorithm would produce a list of wines that would take the person on a tour of different wines, varying in quality throughout the user specified range.

Overall the application is very promising given the amount of time put into development and the goal of developing a software that could actually be used by wine consumers was surely met. With more development time and further optimization, the application would become much more practical and useful for everyday use.