Helping the Homeless
01
April 13, 2019
Group #11
Sam Crawford
Joshua Guinness
Al-Hassan Jasim
Nicholas Mari
SFWR ENG 2XB3 - Software Engineering Practice and Experience: Binding Theory to Practice
Computing and Software
McMaster University

Report Revision History:

April 2nd - Created Half of Title Page of Report
April 3rd - Created Revision Page
April 4th - Wrote Executive summary
April 12th – Worked on Module Decomposition
April 13th – Completed Module Decomposition and Review of Design

Sam Crawford (400129435, crawfs1) - Administrator
Joshua Guinness (400134735, guinnesj) - Project Leader, Lead Tester
Al-Hassan Jasim (400057926, jasim1) - Lead UI Developer
Nicholas Mari (400132494, marin) - Lead Developer

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 [CS] or [SE]-2XB3 project, the reports, presentations, and assignments (not including my name and student number) for future teaching purposes.

Joshua Guinness	April 13, 2019
Sam Crawford	April 13, 2019
Al-Hassan Jasim	April 13, 2019
Nicholas Mari	April 13, 2019

Table of Contents

Contribution Table

Names	Roles	Contributions
Sam Crawford	Administrator	• Kept logs and meeting minutes • Wrote Functional Requirements section of the Requirements Specification • Created Weight module (Weight.java) • Created ADTs: UserInputT.java, UserT.java, AddressT.java, CoolingCentreT.java • Created Master.java and programmed connections between front end and back end, as well as module connections
Joshua Guinness	Project Leader, Lead Tester	• Researched and implemented algorithms that would be best suited for project (MaxPQ.java, TST.java) • Created PowerPoint presentation • Created cover page, revision page, contributions page, executive summary, table of contents for Design Specifications Document • Wrote Non-Functional Requirements section of the Requirements specification • Edited the Requirements Specification • Wrote Javadoc comments for the ADTs
Al-Hassan Jasim	Lead UI Developer	• Wrote Domain section of the Requirements specification • Researched how to create GUIs in Java • Created entire UI for project, including the GUI
Nicholas Mari	Lead Developer	• Wrote Development and Maintenance process section of the Requirements specification • Created ADTs: LocationT.java and ShelterT.java • Created Read module (Read.java) • Solved GUI problems with .jar files and worked to find a solutions by dynamically defining the .classpath • Created UML diagrams for presentation PowerPoint and Design Specifications Document • Worked on the Module Decomposition section of the Design Specifications Document

Executive Summary

Helping the Homeless is a project undertaken with the goal of creating an emergency resource to be used by people struggling with homelessness. Currently, the number of people that are homeless in Toronto is growing (City of Toronto), which is causing the occupancy rate of shelters in Canada to increase (How Many People Are Homeless in Canada).

This product enables users to find the best shelters to go to based on current location, distance, and occupancy rate, as well as the closest cooling centres. This would help solve the problem by getting more people a place to sleep at night.

The data sets used are the daily shelter occupancy data for Toronto from 2017 and 2018, the cooling centre locations in Toronto, and all the municipal addresses in Toronto.

The product consists of a simple GUI that allows a user to find the best shelter based on certain parameters (type of user, type of shelter) or find the closest cooling centre. Both options also have the user to see the directions from their location to the desired shelter or cooling centre in Google Maps.

Sources:

City of Toronto. “Fact Sheet: Why People Are Homeless in Toronto.” City of Toronto, City of Toronto, June 2018, www.toronto.ca/home/media-room/backgrounders-other-resources/fact-sheet-why-people-are-homeless-in-toronto/.

“How Many People Are Homeless in Canada?” Poverty | The Homeless Hub, Canadian Observatory on Homelessness, www.homelesshub.ca/about-homelessness/homelessness-101/how-many-people-are-homeless-canada.

Module Decomposition

NOTE: The data structures used for implementing algorithms (MaxPQ.java, Queue.java, and TST.java) were taken from *Algorithms, Fourth Edition*, by Robert Sedgewick and Kevin Wayne. All documentation for these modules and their methods can be found in the textbook.

Master.java

The purpose of this module is to call the GUI, which inputs the user data, processes it and the datasets, and outputs the best locations. Its public interface consists of one public main method, which is used to call the module responsible for handling the GUI. Since the GUI is dependent on a third party .jar file, the call to the GUI is surrounded in a try-catch statement. If the catch is triggered, the system will output a message to the console, which includes a description of what went wrong, and instructions on how to include the proper .jar file for the user of the program.

Weight.java

The Weight module calculates a weighting for each location. Cooling centres are weighted based on distance to the user, and shelters are weighted based on distance to the user and historical capacity. Having all of the functions for weighting in a module encapsulates the calculations, and provides information hiding, as the ADTs and other modules are blind to how the weighting is performed.

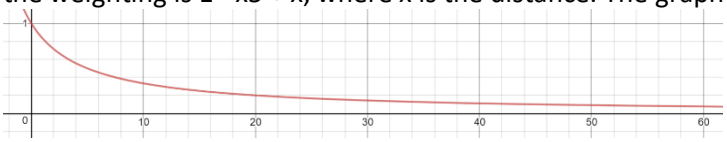
Method Name	Parameters	Return Type
averageOcc	ShelterT shel, int date	int
weightOcc*	ShelterT shel, int date	double
weightDist*	LocationT loc, UserT user	double
calcScore	LocationT loc, UserT user	double
* These are private methods, called by calcScore().		

The two methods in the public interface for this module are averageOcc, which calculates the average occupancy of a shelter, and calcScore, which calculates the weighting of a location from it and the user passed. If the location is a cooling centre, the weight is calculated by weightDist. If it is a shelter, it is calculated with 80% of the final weighting from weightDist() and 20% from weightOcc.

Although averageOcc is called internally by weightOcc, it is also called by the GUI when displaying the best shelter to the output, so it is a public method. The method sums the occupancy of the shelter on the day specified, the three days before, and the three days after (if all indices are in bounds - if the method is called on Jan. 2, only Jan 1 to Jan 5 are summed). The program keeps track of how many days are summed to divide by that number, which results in the average occupancy over the specified time period.

The weighting of a shelter based on historical occupancy is calculated by weightOcc. The current day of the year is passed by calcScore using Java's Calendar library. The average occupancy is calculated by averageOcc. The capacity is obtained from the date passed (arbitrarily) and the final weight is the average occupancy divided by the capacity, subtracted from one. This ensures that shelters with more empty space are given a higher weight.

Distance is weighted for the locations by calculating the distance between the user and the location, based on longitude and latitude. A math function using the Pythagorean Theorem converts the longitudes and latitude into a distance in kilometers. This is then passed through a rational function that maps the distance to a value between zero and one, nonlinearly, so that a shorter distance is weighted much higher than a larger distance would be, proportionately. The function used to map the distance to the weighting is 1 - x5 + x, where x is the distance. The graph is shown to the right:



ADTs

AddressT.java

The address ADT is used to store addresses in a format that can be used with the other modules.

Method Name	Parameters	Return Type
AddressT	String num, String st, double lat, double lon	
getNum	N/A	String
getSt	N/A	String
getLat	N/A	double
getLon	N/A	double

Each address is composed of all its components, a house number, a street name (ie. Berry Creek Dr), and a latitude and longitude. Each field is private and can only be accessed through the corresponding accessor function that is part of the public interface; this is an example of information hiding.

CoolingCentreT.java

This module is an ADT used for storing data related to the cooling centres. It is a subclass of LocationT.

Method Name	Parameters	Return Type
CoolingCentreT	String name, String address, double lat, double lon	
toString	N/A	String

Since a cooling centre doesn't have any additional information than its superclass (LocationT), it doesn't need any more fields or accessors. The reason that a CoolingCentreT exists is to check if the LocationT is a shelter or a cooling centre when printing. The only function it has that is different than LocationT is toString, which outputs a string of the most important information about the cooling centre: its name and address, separated by a pipe.

LocationT.java

This module is an ADT that is used as the superclass for CoolingCentreT and ShelterT. The purpose of having this superclass is to utilize polymorphism, as both CoolingCentreT and ShelterT are both types of locations that utilize many of the same fields.

Method Name	Parameters	Return Type
LocationT	String name, String address	
LocationT	String name, String address, locTypeT type	
LocationT	String name, String address, locTypeT type, double lat, double lon	
getName	N/A	String
getAddress	N/A	String
getLocType	N/A	locTypeT
getScore	N/A	double
getLat	N/A	double
getLon	N/A	double
setScore	double	void
setLon	double	void
setLat	double	void
compareTo	LocationT	int

The public interface for this module consists mostly of accessor and mutator methods which are used to get and set the values of certain fields in the module. There is also a compareTo method used to implement a comparable interface, which compares the locations by their score. The public interface also uses overloading as there are three constructors which are used at different points in the code depending on the parameters passed to the method. Lastly, the public interface also includes an enumerated type called locTypeT which is used to define the type of the location being specified as either a cooling centre or a shelter.

For the private elements of the module, there are no private methods but to follow the principles of information hiding, the fields of the module are private and can only be accessed through the public accessor methods. These fields are the name and address of the location, the type defined by locTypeT, and the score, latitude and longitude of the location.

ShelterT.java

This module is an ADT used for storing data related to the shelters. It is a subclass of LocationT.

Method Name	Parameters	Return Type
ShelterT	shelterResT type, String orgName, String shelterName, String facilityName, String progName, String address	
setCapOcc	int occupancy, int capacity, int year	void
getType	N/A	shelterResT
getTypeString	N/A	String
getOrgName	N/A	String
getFacilityName	N/A	String
getProgName	N/A	String
getOcc2018	int i	int
getOcc2017	int i	int
getCap2018	int i	int
getCap2017	int i	int
isValidType	UserT user	bool
isValidCap	int date	bool
toString	N/A	String

The public interface of this module uses many accessor methods used to access the private fields of the module. The occupancy and capacity accessor methods take an integer value between 0 and 364 to access the data on a given date, since the occupancy and capacity data are stored in arrays. getTypeString also returns the type defined by shelterResT as a String value rather than as a member of the enumerated type. Other than the accessor methods, this module contains a constructor to create an instance of the module as an ADT. The constructor calls the superclass LocationT as a part of polymorphism. The public interface also contains an mutator method called setCapOcc. This method will add capacity and occupancy to the capacity and occupancy arrays respectively, for a given year. 2017 values go in the 2017 versions of the array and 2018 values go in the 2018 versions of the arrays. The module also contains a toString method which converts the data in the ADT into a String representation so data can be printed to the console more easily. Next, the public interface contains two methods that are used to check values. isValidType is used to check if the user's type matches the type of the shelter provided by the ADT. isValidCap is used to check to see if the capacity is greater than 0 for a given date in both 2017 and 2018. Lastly, the public interface includes an enumerated type that is used to define the type of shelter it is. The type is either male, female, youth, coed or family.

The private implementation of this module consists only of private fields, which are private as to preserve the property of information hiding. The fields for this module include the arrays for the 2017 and 2018 capacities and occupancies, the names and addresses of the shelter, the type of shelter and two counter values to keep track of the indices of the capacity and occupancy arrays.

UserT.java

The User ADT allows for storing the required information about the user to allocate them to the best location.

Method Name	Parameters	Return Type
UserT	UserResT type, double lat, double lon	
UserT	double lat, double lon	
getResType	N/A	UserResT
getLat	N/A	double
getLon	N/A	double

The public interface consists of an overloaded constructor (an example of polymorphism) and public accessors for private fields (an example of information hiding). The first constructor is for use when looking for a shelter, as it takes in the type of the user, to be matched with a shelter. However, both shelters and cooling centres require the location of the user for calculating the distance between the location and user, so these values appear in both constructors. A UserT is comprised of an enumerated resident type (which denotes which shelters the user would want to stay at: male only, male with coed, etc.) if being used with a shelter, and a latitude and longitude, and the accessors return their respective values.

Input/Output

FindCoolingCentre.java

This module finds the best cooling centres based on the inputted user location.

Method Name	Parameters	Return Type
open	N/A	void
createContents*	N/A	void
*This is a protected method used by open()		

The public interface of the module is open, which calls createContents and displays the contents. The protected method createContents initializes all the components, such as buttons and text boxes, and sets their attributes, such as text and size. The method creates a textbox for the user to input their address, as well as a button to find the nearest cooling centre. Once the button is clicked, the method uses the address entered in the textbox to create a MaxPQ of cooling centres, by reading the cooling centre data, building a user type, and weighting each cooling centre.

FindShelter.java

This module finds the best shelters based on the inputted user location and user type.

Method Name	Parameters	Return Type
open	N/A	void
createContents*	N/A	void
getUserType**	N/A	UserResT
concatenate**	ShelterT[] a, ShelterT[] b	ShelterT[]
*This is a protected method used by open(). ** These are private methods used by createContents.		

The public interface of the module is open, which calls createContents and displays the contents. The protected method createContents initializes all the components, such as buttons and text boxes, and sets their attributes, such as text and size. The method creates a textbox for the user to input their address, some radio buttons to specify the user's attributes, and a button to find the nearest shelter. Once the button is clicked, the method uses the address entered in the textbox to create a MaxPQ of the shelters of the correct type, by reading the shelter data, building a user type, and weighting each shelter.

The two private methods called by createContents are used to simplify the code. The method getUserType returns a UserResT from the current private booleans representing the user's type, which is used to find a valid shelter by type. The trivial concatenate takes two arrays of ShelterT and outputs an array of ShelterT containing every shelter in each individual array. This allows the male/female and coed arrays to be combined if necessary, to form one MaxPQ for accessing the best shelter.

MainWindow.java

This module allows the user to choose whether they want to find the closest shelter to them or whether they want to find the closest cooling shelter. This can be done by utilizing two buttons that when clicked opens a different user interface module. In this module, no new methods were added. Once one of the buttons is clicked the corresponding display.

OutputWindow.java

This module outputs the best locations to the user and allows them to scroll through the list and open any location in Google Maps, with directions to it from their current location. It also prints the name of the location to the console.

Method Name	Parameters	Return Type
open	N/A	void
createContents*	N/A	void
*This is a protected method used by open()		

The public interface of the module is open, which calls createContents and displays the contents. The protected method createContents initializes all the components, such as buttons and text boxes, and sets their attributes, such as text and size. The method creates a button to display the next location and another to display directions to the currently displayed location in Google Maps.

Read.java

This module is designed to read all the data from the datasets and return the data organised into different data structures. The program was decomposed into this module in order to preserve encapsulation. This is because this module contains all the methods pertaining to the reading of data from the datasets and the purpose of encapsulation is to package like methods and functions together.

Method Name	Parameters	Return Type
readShelterData	N/A	ShelterT[][]
readCoolingData	N/A	CoolingCentreT[]
readAddressData	N/A	TST<AddressT>

contains*	String[] vals, ShelterT shelter	bool
addToList*	ArrayList<ShelterT> arr, String[] vals, shelterResT type, int occ, int cap, int year	void
*These private methods are used by readShelterData()		

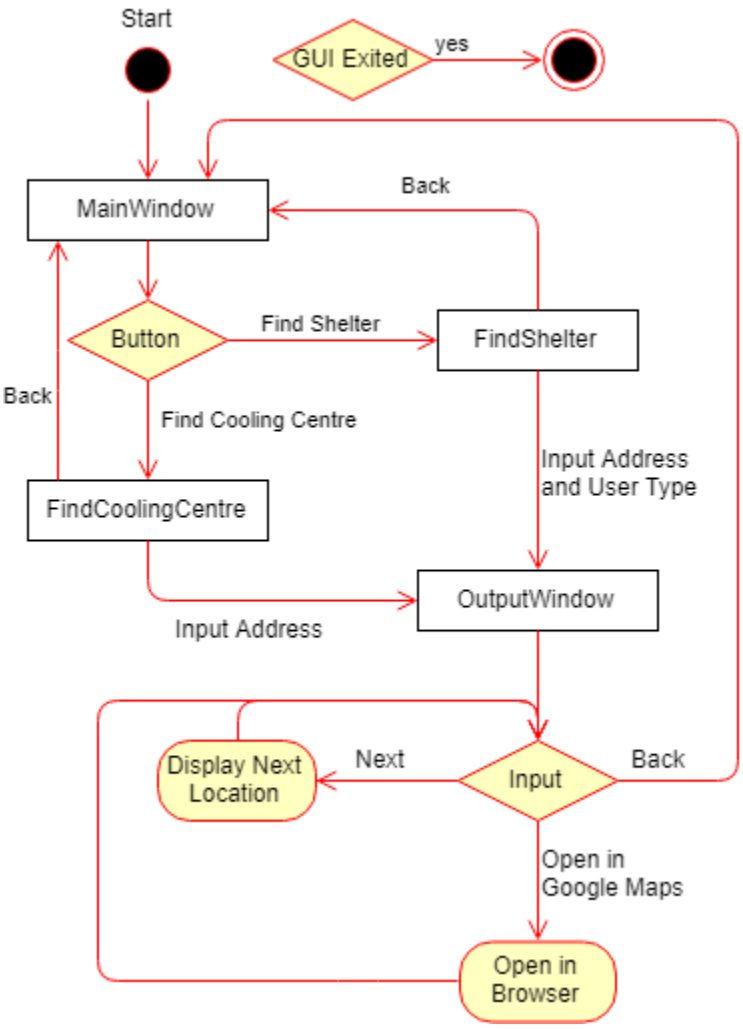
The public interface for this module contains three methods: readShelterData, readCoolingData, and readAddressData. The readShelterData method reads and organizes the data from both the 2018 and 2017 versions of the homeless shelter occupancy datasets. This method returns a 2D array of type ShelterT, which is the ADT designed to store the data of a single shelter. This method reads the data using a scanner object and system.in. Since the data is in a .csv file, all the data is comma separated with new lines for each row in the .csv file. The method reads each line, except for the first which contains the headers, and stores them in a string variable. The method then splits this string at each instance of a comma and stores this in a string array. From this string array we take the organization name, shelter name, address, facility name, program name, shelter type, occupancy data and capacity data, which occur at indexes 1, 2, 3, 7, 8, 9, 10 and 11 respectively, and send it to a private method which creates a new instance of the ShelterT ADT and adds it to an ArrayList. The ArrayList that the ADT is stored in depends on the type of the shelter. If the shelter is male only, then the data is stored into an Arraylist of all the male shelters. This is the same for shelters that are female only, co-ed, families and youth as well. This process will continue for the entire csv file for both the 2018 and 2017 datasets. Once all the data has been stored in one of the five ArrayLists, they are converted into normal arrays and then combined into one 2D array with each index of the 2D array containing one of the arrays corresponding to the type of shelter it is.

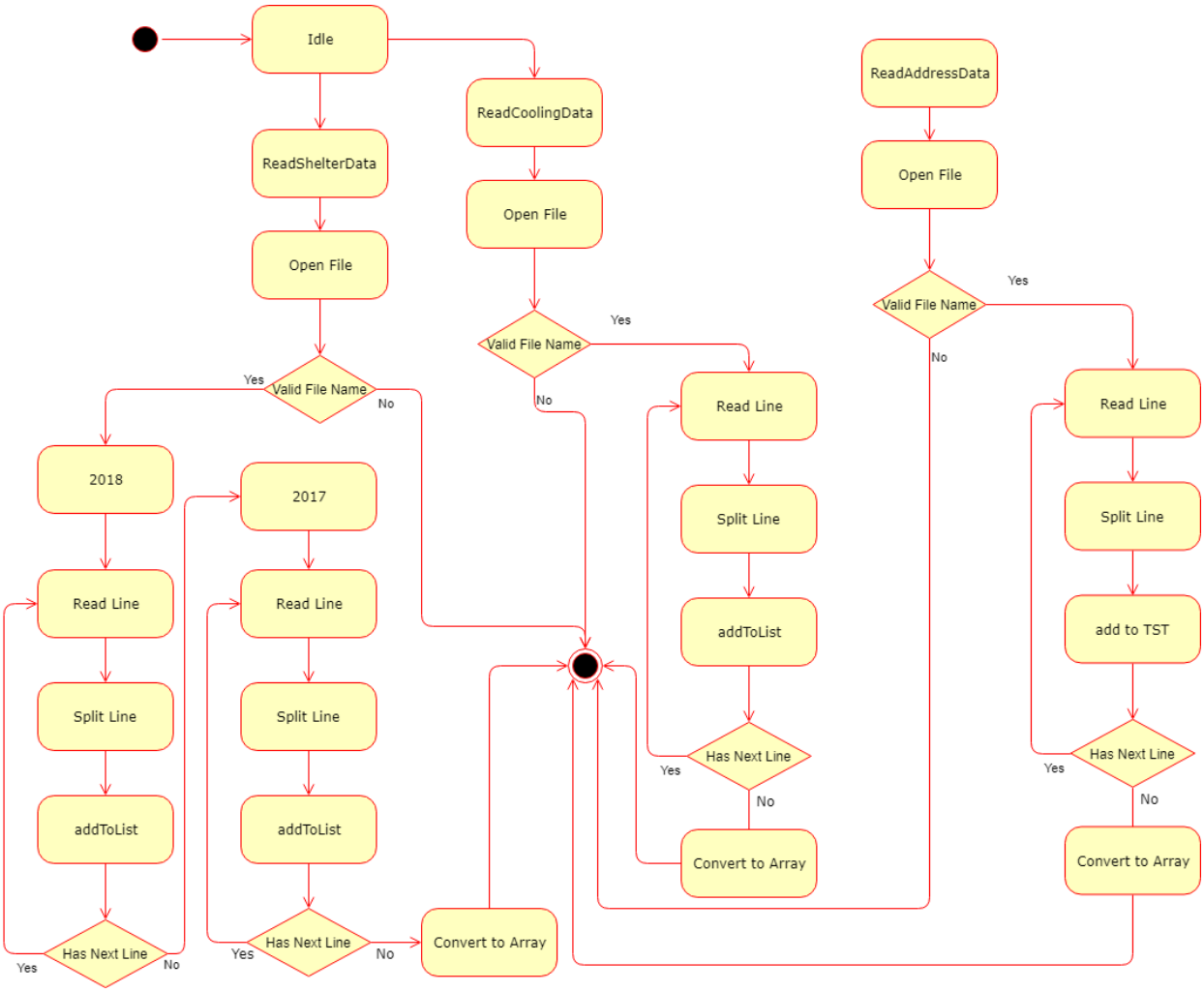
The readCoolingData method works in a similar manner to readShelterData, by reading in a csv file of data using a scanner object. However, unlike readShelterData, this method returns a 1D array of type CoolingCentreT. This method reads the name, address, latitude and longitude from the .csv file and creates an ADT of type CoolingCentreT, which is then stored in an ArrayList of the same type. Once all the data has been read and stored, the ArrayList is converted into an array and returned to the user.

This pattern continues with readAddressData, which is used to read data about the municipal address points in Toronto and store it in a TST. This method uses a scanner object to read the data from a .csv file. The program takes the address number and name, and the latitude and longitude and creates an ADT of type AddressT. The AddressT ADT is then put into a TST of type AddressT.

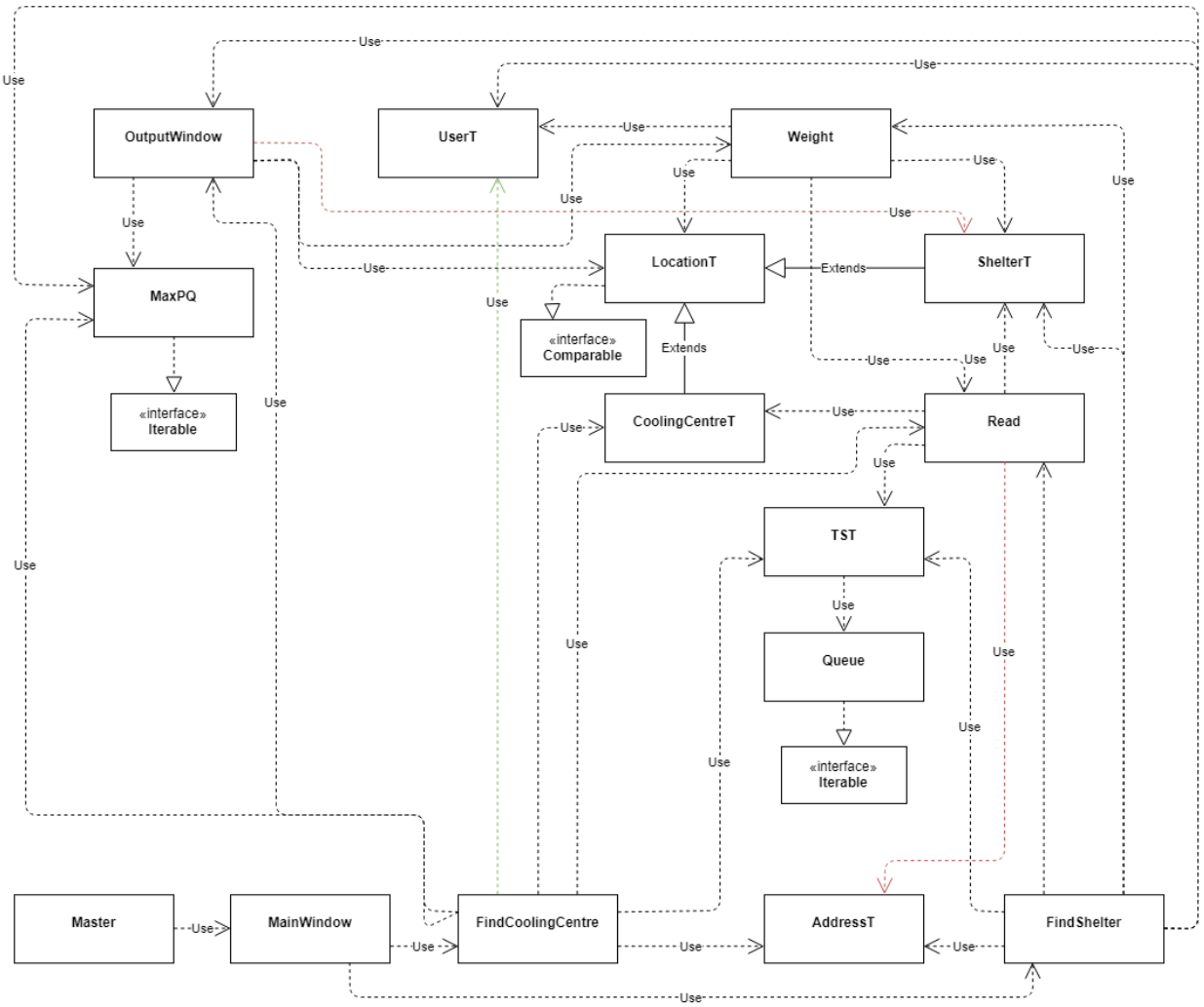
In this module, there are two private methods used to aid in the implementation of our functional requirements. The first of these two methods is contains. This method takes a shelter of type ShelterT and a String array and then returns a boolean value. These are used in order to check if the values found in the string array are the same as the values in the shelter provided by the shelterT parameter. If the names, and addresses are the same, then the method returns true, otherwise it returns false. The next method is addToList, which is a private method used by readShelterData to add capacity and occupancy data to an ADT stored in the ArrayList or add a new ADT to the ArrayList. The method accepts an ArrayList of type ShelterT, an array of string values, the occupancy and capacity data of the shelter and the type of the shelter. This method will check if the shelter defined by the values in the string array math a shelter already found in the ArrayList using the contains() method discussed above. If contains returns true, the occupancy and capacity data are added the ADT that matches, and the method returns control to where it was called. Otherwise, the method will create a new ADT and store it in the ArrayList.

UML State Machine Diagrams




Idle
ReadShelterData
Open File
Valid File Name
Yes
No
2018
Read Line
Split Line
addToList
Has Next Line
Yes
No
2017
Read Line
Split Line
addToList
Has Next Line
Yes
No
Convert to Array
ReadCoolingData
Open File
Valid File Name
Yes
No
Read Line
Split Line
addToList
Has Next Line
Yes
No
Convert to Array
ReadAddressData
Open File
Valid File Name
Yes
No
Read Line
Split Line
add to TST
Has Next Line
Yes
No
Convert to Array

UML Uses Relationship Diagram



Internal Review and Evaluation of Design

The use of ADTs allows for encapsulation and information hiding, by grouping similar functions together and allowing modules to only obtain the data they need from the ADTs. Overloading constructors, as per the principle of polymorphism, allows different “flavours” of the ADT to be built for different applications, while still maintaining the common methods and functionality. Polymorphism is also demonstrated by the inheritance of LocationT by ShelterT and CoolingCentreT, which prevents the duplication of code for common tasks.

However, our design is not without its faults. The Master module only calls the GUI, which then does all of the reading and processing of the data. This violates information hiding, since the GUI should only be responsible for displaying data, not generating it, but because of how we decided to implement the GUI early on, the easiest way to implement our program is by doing everything internally. The input from the GUI goes right to being processed, instead of being returned to the Master module, and the output from the processing goes straight to being displayed on the GUI.

As described above, this is obviously not ideal. The design pattern that should have been used for this is the model-view-controller design pattern. In this pattern, we would have a module to control everything that would happen, in our case Master, modules to model the program, and a view module. Our design was loosely based off of this but due to time constraints and the type of GUI implemented, its implementation was less effective than it should have been.

Upon further review, the type of GUI chosen was not the best one to implement for the project. This GUI requires certain files that differ between various operating systems, which makes it extremely difficult to develop a cross-platform product. Separating view and control when implementing this GUI was also burdensome.