LiveLong

Version 1.4

CS 2XB3
Computer Science Practice and Experience: Binding Theory to Practice
Lab Section: L02
Group: 08

Kenneth Mak - 001318946

April 12, 2020

Revision History

Revision	Date	Author(s)	Description
1.0	08.04.2020	KM	Document created
1.1	09.04.2020	KM	Figures added
1.2	09.04.2020	KM	Description of implementation added, edits to figures
1.3	11.04.2020	KM	Added Internal Review of design, edits to phrasing in other areas
1.4	12.04.2020	KM	Document finalized

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

Contents

1 Contributions

As this project was completed by a single member, the Role column is deemed unnecessary. Instead the contributions will list the amount of work completed.

Name	**Contributions**
Kenneth Mak	Research of Valid Projects Wrote Project Proposal Project Presentation Wrote Requirements Specifications Designed and implemented DataReader class to read from CSV files Implemented DataWriter class to write object data to CSV files Geocoded the locations of all NursingHomes into a dataset NursingHome ADT NHGeoInfo ADT Deficiency ADT Implementation of longitude-latitude distance calculations Implementation of geocoding an address during runtime using OSM Implemented ability to change user location during runtime Implementation of QuickSort algorithm for arrays and lists Implementation of Sequential Search Symbol Table Implementation of Separate-Chaining Hash Table for efficient Sort algorithms Added sorting order to NursingHome and NHGeoInfo ADT Implemention of search filter to return list of objects from the database Created interactive menu with the command line interface Wrote TextParser to respond to commands Implemented Application run-loop and basic logic Designed text UI and viewer Implemented ability to open GoogleMaps on desktop browser Handled errors and bugs Wrote documents and specifications

2 Executive Summary

LiveLong is an application that quickly and accurately delivers information about nursing homes in the U.S. with official government datasets provided by Medicare. Clients can quickly search through the official datasets to get a quick and informed look at prospective nursing homes. Clients may modify a search filter to get better results that fulfil their requirements, such as selecting nursing homes that are 'Special Focus', or only selecting nursing homes with a rating over 3.

Results are presented based on overall ratings and distance to the client. The ratings of each nursing home are conducted with current and previous residents. These ratings include areas concerning long-term and short-term stay experiences, the quality of the services, and the quality of the staff.

The application also highlights any warnings and active deficiencies the nursing home has, such as having a history of abuse cases. Active deficiencies, should there be any, will notify the clients of its severity and the duration of when it was first assigned, and whether it has been resolved or not.

3 Module and Class Overview

3.1 Module Decomposition

The application was implemented with a Model-View-Controller design pattern. The application is separated into four different packages.

1. **classes** - Contains all ADT classes
2. **data** - The Model, database and file parsing classes
3. **app** - The Main application, View and Controller classes
4. **util** - Utilities module providing access to data structures, algorithms, and access to OSM & GoogleMaps functionalities

Refer to Figure 1 for visual representation of classes and their relationships.

3.1.1 classes package

Classes placed in here are used to store information parsed from a Dataset into an object.

- **NursingHome.java** ADT stores information about a Nursing Home. Each NursingHome has a unique ID
- **NHGeoInfo.java** ADT stores geographical information about a Nursing Home. Corresponds to the NursingHome object with the same ID.
- **NHPair.java** ADT stores a corresponding **NursingHome** and **NHGeoInfo**. Used for passing search results
- **Deficiency.java** ADT stores information about a Deficiency that was assigned to a NursingHome. Refers to a specific NursingHome by ID.
- **FireSafetyDeficiency.java** ADT is a subclass of Deficiency
- **HealthDeficiency.java** ADT is a subclass of Deficiency

3.1.2 data package

Classes are used to parse information from CSV files and store them in memory to be interacted with. It also provides the ability to write compressed CSV files to be used for initialization instead of the original datasets, reducing initialization times.

- **Database.java** is the Model of the MVC design pattern. All ADTs from classes constructed from parsed datasets are stored here.
- **DataReader.java** is used to read information from datasets and construct the appropriate ADTs
- **DataWriter.java** is used to write ADT information into a CSV format such that the same ADTs can be constructed with the CSV

3.1.3 app package

Classes placed in here are used to parse user response and pull information from the Database, which is located in the **data** package.

- **LiveLong.java** is the main application loop
- **Controller.java** is used to send requests to the Database and receive the result of searches. Passes results to **TextParser**
- **TextParser** is the View of the MVC pattern. Displays the UI in the command line and parses user response into commands for the Controller to send.
- **State.java** is an enum denoting the current state of the application (I.e. Menu, Settings, Exiting, etc.)

3.1.4 util package

Classes here provide common utility functions, algorithms, or data structures to be used by other classes. Not specific to this LiveLong application.

- **GoogleMapUtils.java** provides a method to open GoogleMaps in desktop browser with an origin and destination
- **OpenStreetMapUtils.java** provides a method to return the geocoded location details of an address
- **QuickSort.java** provides QuickSort algorithm implementation for arrays and lists
- **SeparateChainingHashST.java** provides a HashTable for efficient insert and search operations. Uses SequentialSearchST.
- **SequentialSearchST.java** provides a SymbolTable list for quick insert and search operations.

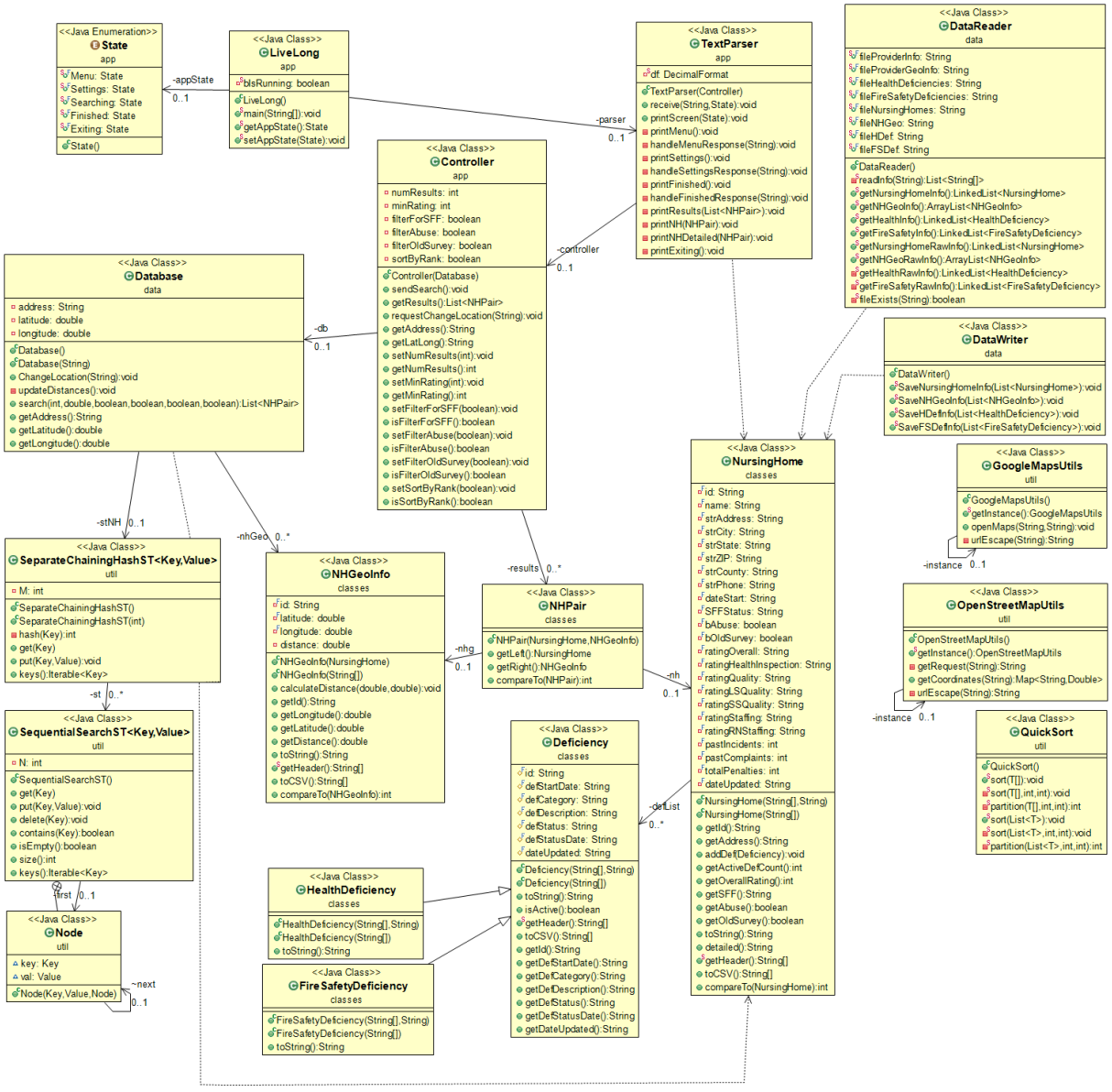


Figure 1: UML Class diagram showing a static representation of LiveLong application classes and relationships

Both **SequentialChainingHashST** and **SequentialSearchST** were created from Lab 9 of CS2XB3.

3.1.5 Explanation

The usage of an MVC design pattern enforces the Separation of Concerns design principle. Changes to one module or class should not affect another module that they both needed be modified. This allows for faster and more efficient modifications to the application to be carried out in the future.

For example, as a possible change, say the application needs to change from a CLI program to a visual GUI. The developer would not need to make changes to how the Database or Controller modules work. All they would have to do is develop a new class to be used in place of TextParser to act as the View of the MVC design pattern. This new class would instead generate a GUI, but still send parsed user interactions to the unchanged Controller module.

Other possible changes could include how CSV files may have their structure changed, or if better algorithms need to be implemented to account for scalability. These changes can then be fixed by only modifying their specific code in **DataReader** and in **SequentialSearchHashST**

3.2 Classes Package Overview

3.2.1 NursingHome ADT

public class NursingHome

	NursingHome(String[])	Constructs an immutable NursingHome object from a String array, usually parsed from the CSV file
String	getID()	Returns the ID of this NursingHome
String	getAddress()	Returns the Address of this NursingHome
void	addDef()	Adds a Deficiency to this NursingHome
int	getActiveDefCount()	Counts the number of active deficiencies
int	getOverallRating()	Gets the overall rating
String	getSFF()	Gets the Special Focus status
boolean	getAbuse()	Returns Abuse variable
boolean	getOldSurvey()	Returns OldSurvey variable
String	toString()	Returns a String showing only the important values
String	detailed()	Returns a String with all the values
String[]	toCSV()	Returns all values in state variables

3.2.2 NHGeoInfo ADT

public class NHGeoInfo

	NHGeoInfo(String[])	Constructs a NHGeoInfo object from a String array, usually parsed from the CSV file
void	calculateDistance ($lat2$, $lon2$)	Calculates and sets the distance value of this object to the distance between this latitude and longitude with target $lat2$ and $lon2$
String	getID()	Returns the NursingHome ID that this NHGeoInfo refers to
double	getLatitude()	Returns the latitude value
double	getLongitude()	Returns the longitude value
double	getDistance()	Returns the distance between this object and a target location
String[]	toCSV()	Returns all values in state variables

3.2.3 NHPair ADT

public class NHPair

	NHPair (NursingHome, NHGeoInfo)	Constructs a NHPair object with the two objects, such that both their IDs are the same
NursingHome	getLeft()	Returns the NursingHome object
NHGeoInfo	getRight()	Returns the NHGeoInfo object

3.2.4 Deficiency ADT

public class Deficiency

	Deficiency(String[])	Constructs an immutable Deficiency object from a String array, usually parsed from the CSV file
boolean	isActive()	Check the status of deficiency and the status date
String	getID()	Returns the NursingHome ID that this Deficiency refers to
String	getDefStartDate()	Returns when this deficiency was assigned
String	getDefCategory()	Returns the category of this deficiency
String	getDefDescription()	Returns the description of this deficiency
String	getDefStatus()	Returns the status of this deficiency
String	getDefStatusDate()	Returns the date of when the status changed
String[]	toCSV()	Returns all values in state variables

3.2.5 FireSafetyDeficiency ADT

public class FireSafetyDeficiency extends Deficiency

	FireSafetyDeficiency (String[])	Constructs an immutable FireSafetyDeficiency object from a String array, usually parsed from the CSV file
String	toString()	Returns a string detailing this Fire Safety Deficiency

3.2.6 HealthDeficiency ADT

public class HealthDeficiency extends Deficiency

	HealthDeficiency (String[])	Constructs an immutable HealthDeficiency object from a String array, usually parsed from the CSV file
String	toString()	Returns a string detailing this Health Deficiency

3.3 Data Package Overview

3.3.1 Database

public class Database

	Database (String)	Sets user location to string and get geocode of location. Parses the information from the dataset into data structures with DataReader. NursingHome objects are placed into a Hash table. Deficiencies are added to the corresponding NursingHome object by ID. NHGeoInfo objects have their distance values recalculated to the user location.
void	ChangeLocation (String)	Attempts to change the user location to new parameter. If Geocoding the new address is successful, save new location and update all NHGeoInfo objects with new location.
seq of NHPair	search(int, int, bool, bool, ...)	Returns a list of NHPair objects with NursingHome objects that satisfy the search criteria in the parameters
double	getAddress()	Returns the address/user location
double	getLatitude()	Returns the latitude value of address
double	getLongitude()	Returns the longitude value of address

3.3.2 DataReader

public class DataReader

static (seq of NursingHome)	getNursingHomeInfo()	Returns a list of NursingHome objects constructed from the CSV dataset
static (seq of NHGeoInfo)	getNHGeoInfo()	Returns a list of NHGeoInfo objects constructed from the CSV dataset
static (seq of FireSafetyDeficiency)	getFireSafetyInfo()	Returns a list of FireSafetyDeficiency objects constructed from the CSV dataset
static (seq of HealthDeficiency)	getHealthInfo()	Returns a list of HealthDeficiency objects constructed from the CSV dataset

3.3.3 DataWriter

public class DataWriter

static void	SaveNursingHomeInfo (seq of NursingHome)	Saves a list of NursingHome objects into a CSV
static void	SaveNHGeoInfo (seq of NHGeoInfo)	Saves a list of NHGeoInfo objects into a CSV
static void	SaveFireSafetyInfo (seq of FireSafetyDeficiency)	Saves a list of FireSafetyDeficiency objects into a CSV
static void	SaveHealthInfo (seq of HealthDeficiency)	Saves a list of HealthDeficiency objects into a CSV

3.4 App Package Overview

3.4.1 LiveLong

public class LiveLong

static void	main(String[])	Initialize the **Database**, **Controller**, and **TextParser** classes. Set application state to *State.Menu*. Start application loop. In each loop, request user input. Upon receiving user input, send to TextParser the response and current application state. End loop when application state changes to *State.Exiting*.
static State	getAppState (String)	Returns the current application state
static void	setAppState (State)	Changes the application state to parameter. Prompts **TextParser** to printScreen(State)

3.4.2 State enum

public enum State

State Enum	
Menu	While in Menu state, user can choose to start a search, go to Settings state, or exit
Settings	While in Settings state, user can choose to modify their search results and current location, or return to Menu state.
Searching	While in Searching state, application is requesting results from Database based on search filter. No user input allowed until Database finishes search
Finished	While in Finished state, results of search are shown on screen. User can choose to examine a result and open them in Google Maps. User can return to Menu state
Exiting	While in Exiting state, application is shutting down. No user input allowed.

3.4.3 Controller

public class Controller

	Controller (Database)	Initialize the **Controller** with a reference to the Database
void	sendSearch (int, int, bool, bool, ...)	Send parameters to db.search() function, and store the results in local variable
seq of NHPair	getResults()	Return the list of NHPair received after sendSearch() method
void	requestChangeLocation (String)	Send a prompt to Database to change locations
String	getAddress ()	Get the current user location from the Database object
void	setNumResults(int)	Change search filter to return only x amount of results
int	getNumResults()	Get current maximum number of results
void	setMinRating(int)	Change search filter to return objects with rating over x
int	getMinRating()	Get current minimum rating in filter
void	setFilterForSFF (bool)	Change search filter SFF to bool
boolean	isFilterForSFF()	Return true if search filtering for SFF Nursing Homes
void	setFilterAbuse(bool)	Change search filter for Abuse flags to bool
boolean	isFilterAbuse()	Return true if search filtering out Abuse flags
void	setFilterOldSurvey (bool)	Change search filter for OldSurvey flags to bool
boolean	isFilterOldSurvey()	Return true if search filtering out OldSurvey flags
void	setSortByRank(bool)	Change if results should be sorted by ratings first
boolean	isSortByRank()	Return true if results are returned sorted by ratings first, then distance

3.4.4 TextParser

public class TextParser

	TextParser(Controller)	Initialize the **TextParser** with a reference to the Controller
void	receive(String, State)	Handle the String response of user based on the current State, and send parsed commands to Controller
void	printScreen(State)	Prints the screen corresponding to the given State in the command line

3.5 Util Package Overview

3.5.1 GoogleMapsUtils

public class GoogleMapsUtils

	GoogleMapsUtils()	Create singleton instance of self
static GoogleMapsUtils	getInstance()	Returns the singleton instance
void	openMaps (String, String)	Opens GoogleMaps in default desktop browser with the origin and destination set to the provided locations

3.5.2 OpenStreetMapUtils

public class OpenStreetMapUtils

	OpenStreetMapUtils()	Create singleton instance of self
static OpenStreetMapUtils	getInstance()	Returns the singleton instance
Map ⟨String, double⟩	getCoordinates(String)	Attempts to geocode the given address and return the latitude and longitude

3.5.3 QuickSort

public class QuickSort

void	sort(T[ ])	Sorts the given array using QuickSort
void	sort(List<T>)	Sorts the given list using QuickSort

SeparateChainingHashST and **SequentialSearchST** classes are pulled from Lab 9 of CS2XB3.

3.5.4 SeparateChainingHashST

public class SeparateChainingHashST

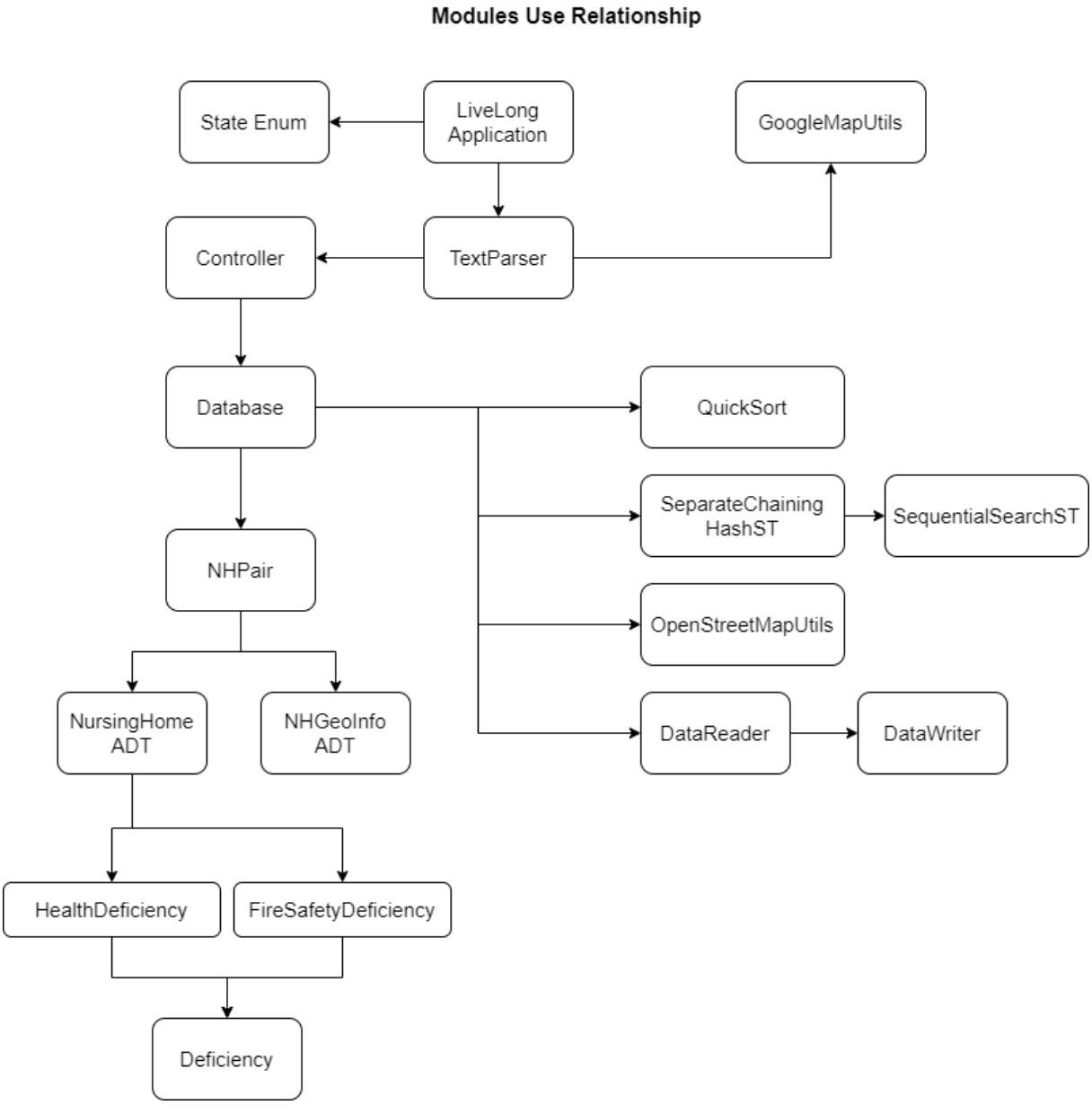
	SeparateChainingHashST(int)	Constructs a hash table with size of int
Value	get(Key)	Searches for the Key in hash table, and returns the Value associated with it
void	put(Key, Value)	Inserts the Key and Value in hash table. If key exists, replace previous value with new Value
Iterable <Key>	keys()	Returns the set of all keys in hash table

3.5.5 SequentialSearchST

public class SequentialSearchST

	SequentialSearchST()	Initializes an empty linked list of nodes
Value	get(Key)	Searches for the Key in linked list, and returns the Value associated with it
void	put(Key, Value)	Inserts the Key and Value in list. If key exists, replace previous value with new Value
void	delete(Key)	Removes the target Key and associated Value from list
boolean	contains(Key)	Return true if key is found in list
boolean	isEmpty()	Return true if size() is 0
int	size()	Returns size of list
Iterable <Key>	keys()	Returns the set of all keys in list

4 View of the Uses relationship



5 Tracing Back to Requirements

Functional Requirements	Modules Handling Requirements
NursingHome ADT	**NursingHome**, **Deficiency**
NHGeoInfo ADT	**NHGeoInfo**, **DataReader**, **OpenStreetMapUtils**
Generic Deficiency ADT	**Deficiency**, **DataReader**
FireSafetyDeficiency ADT	**FireSafetyDeficiency**, **Deficiency**, **DataReader**
HealthDeficiency ADT	**HealthDeficiency**, **Deficiency**, **DataReader**
NHDatabase	**Database**, **DataReader**, **DataWriter**
CSVReader	**DataReader**, **DataWriter**
Menu	**Controller**, **TextParser**, **LiveLong**, **State**

NF Requirements	Modules Handling Requirements
Reliability	**Database**, **DataReader**, **SequentialChainingHashST**, **QuickSort**
Accuracy of Results	**Database**, **DataReader**, **SequentialChainingHashST**, **QuickSort**
Performance	**DataReader**, **DataWriter**, **QuickSort**, **SequentialChainingHashST**
Usability	**LiveLong**, **GoogleMapsUtils**, **TextParser**, **Controller**
Scalability	**DataReader**, **DataWriter**, **QuickSort**, **SequentialChainingHashST**

6 Implementation Details

6.1 classes package

6.1.1 NursingHome.java

An immutable object that is constructed directly from a row in the CSV file. Its variables are all used to store information regarding a Nursing Home in the US Medicare dataset.

Additionally, there is a variable that holds a List of Deficiency objects to store FireSafetyDeficiency and HealthDeficiency objects that refer to this NursingHome. There is a method to return the number of active deficiencies in this list.

There is a method used to return all values, except for the **List<Deficiency>** variable, as a String array. This is used to allow the writing of data into a compressed CSV file that only has the important information, which can then be used to initialize the database faster.

Finally, there is a simplified toString() method printing out a summarized version of the values, and a detailed() method returning a string with all the details.

6.1.2 NHGeoInfo.java

An object constructed from the dataset pertaining to the geological information of a NursingHome. It contains the **id**, **latitude**, and **longitude** values, which are immutable. The last variable ***distance*** is calculated using the given lat and long values, and taking another location's values as the target.

The **distance** variable is mutable to allow for the application to change the user's location during runtime.

There is a method used to return all values as a String array. This is used to allow the writing of data into a compressed CSV file that only has the important information, which can then be used to initialize the database faster.

6.1.3 NHPair.java

NHPair is used to create a pair of **NursingHome** and **NHGeoInfo** objects that both have the same ID value. Used primarily when returning the results of a search filter, and provides both the values and the geological information of a Nursing Home. Additionally,

it allows for a unique sort order such that a list of NHPair is sorted by NursingHome ratings first, then by NHGeoInfo's distance value.

6.1.4 Deficiency.java

An immutable object constructed from the datasets referring to either FireSafetyDeficiencies or HealthDeficiencies datasets. Its variables are used to store information parsed from the CSV file.

There is a method to check whether this Deficiency is still active based on the Deficiency Status and Status Date. This is used when determining if this deficiency should be shown to the user (I.e. if no longer active, do not show)

Finally, there is a method used to return all values as a String array. This is used to allow the writing of data into a compressed CSV file that only has the important information, which can then be used to initialize the database faster.

6.1.5 FireSafetyDeficiency and HealthDeficiency.java

No private entities. Subclasses of Deficiency.java. Only difference between the two is their **toString()** methods

6.2 data package

6.2.1 Database

Contains the following private variables:

- String **address** : storing location of the user
- double **latitude** : storing latitude value of address
- double **longitude** : storing longitude value of address
- SeparateChainingHashST<String, NursingHome> **stNH**:
  storing NursingHome objects in hash table using NursingHome.getId() as Key.
- List<NHGeoInfo> **nhGeo** : storing NHGeoInfo objects as a list. Used for applying QuickSort

It contains one private method "**updateDistances()**" to go through all NHGeoInfo objects in **nhGeo** and prompts them to recalculate their **distance** variable with the latitude and longitude values. This is called once at initialization, and whenever the database is prompted to change the user's location.

Once updated, it does a QuickSort on nhGeo to re-sort the list from lowest distance to highest, so that when answering a search request it needs to simply start from the first index of **nhGeo**.

Before returning the **results** of a search as a **List<NHPair>**, if the search filter requested the results ordered by ratings, it will do a QuickSort on the results list before returning. This then has the list ordered by ratings instead of distance.

Refer to Figure 2 for the Database State UML.

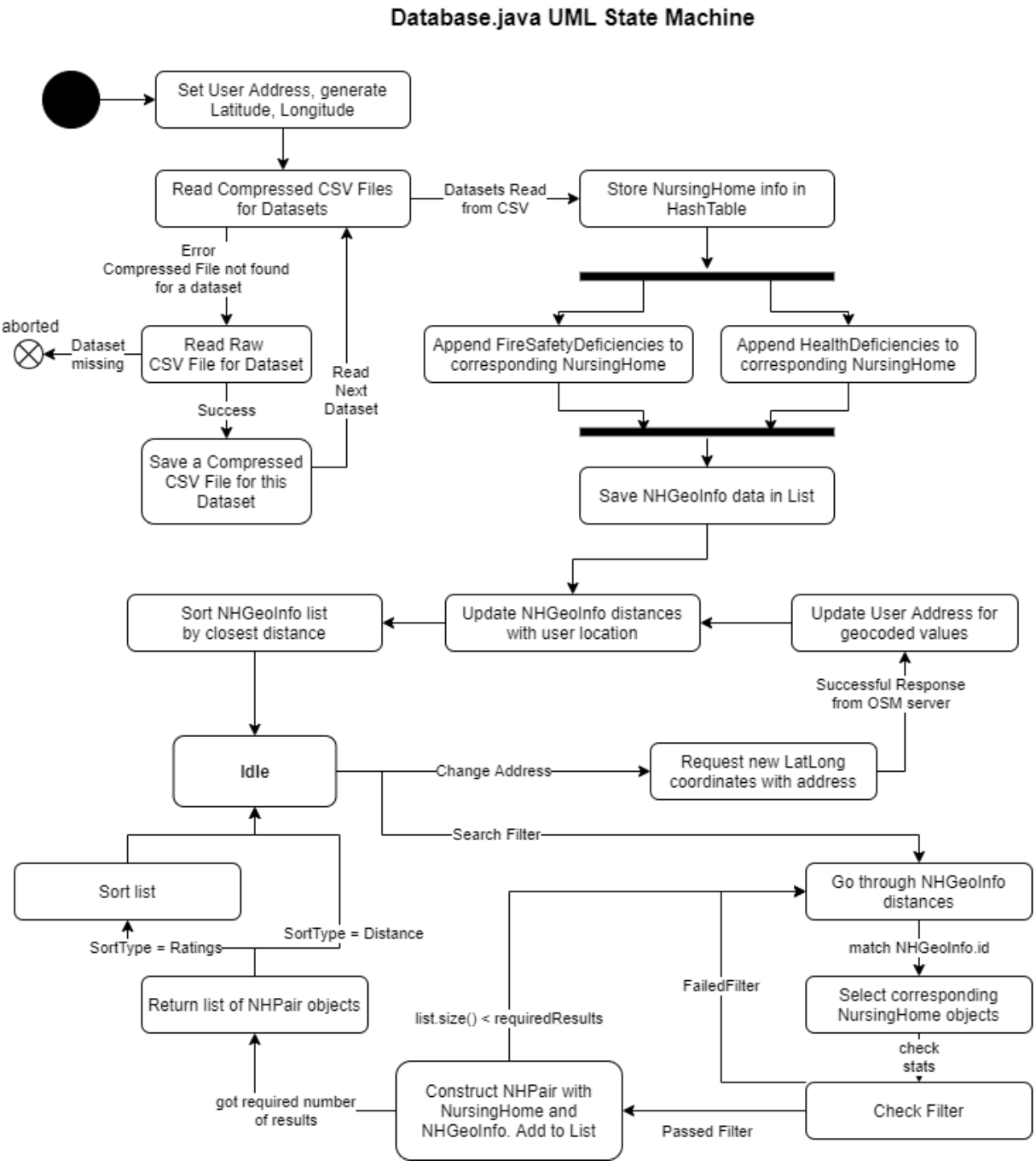


Figure 2: UML State Machine diagram for **Database.java**. This class is used to parse and store the information from the dataset CSV files into appropriate data structures. It is constructed with the main goal of being able to provide quick and efficient operations on the data to account for scalability with growing datasets.

6.2.2 DataReader

Has private methods to check if a file exists before trying to read. Each public read method has a flow to go through:

1. Attempt to read compressed CSV file
    (a) If failed (I.e. File not found), try to read Raw CSV file.
2. Parse the List<String[]> from the CSV
3. Construct the object from the String[] and store in a List<Object>
    (a) If reading Raw CSV file, use DataWrite to save this List<Object> as a compressed CSV version for future use.
4. Return the List<Object>. End.

If both the compressed and raw versions of the datasets are not found, throw an exception and stop the application.

6.2.3 DataWriter

No private methods. Used to simply write a List<String[]> into a CSV file delimited by commas. Generally, this CSV file is a more compressed version of the raw dataset, since it does not include unused columns or empty cells.

6.3 app package

6.4 LiveLong

Contains a private **State** enum variable to control what the user should see and what interaction they can do.

Contains a private variable to reference a **TextParser** object, which it where it sends the user's input in the Command Line along with the current application state.

Uses a while loop to continuously wait for a user's input to send to the TextParser. Until the application state is changed to Exiting, this application loop will repeat.

The **setAppState(State)** and **getAppState()** are 'static', such that they can be accessed from other classes.

Upon using **setAppState(State)**, tell **TextParser** to print the screen corresponding to the current State

Refer to Figure 3 for a LiveLong Application State UML.

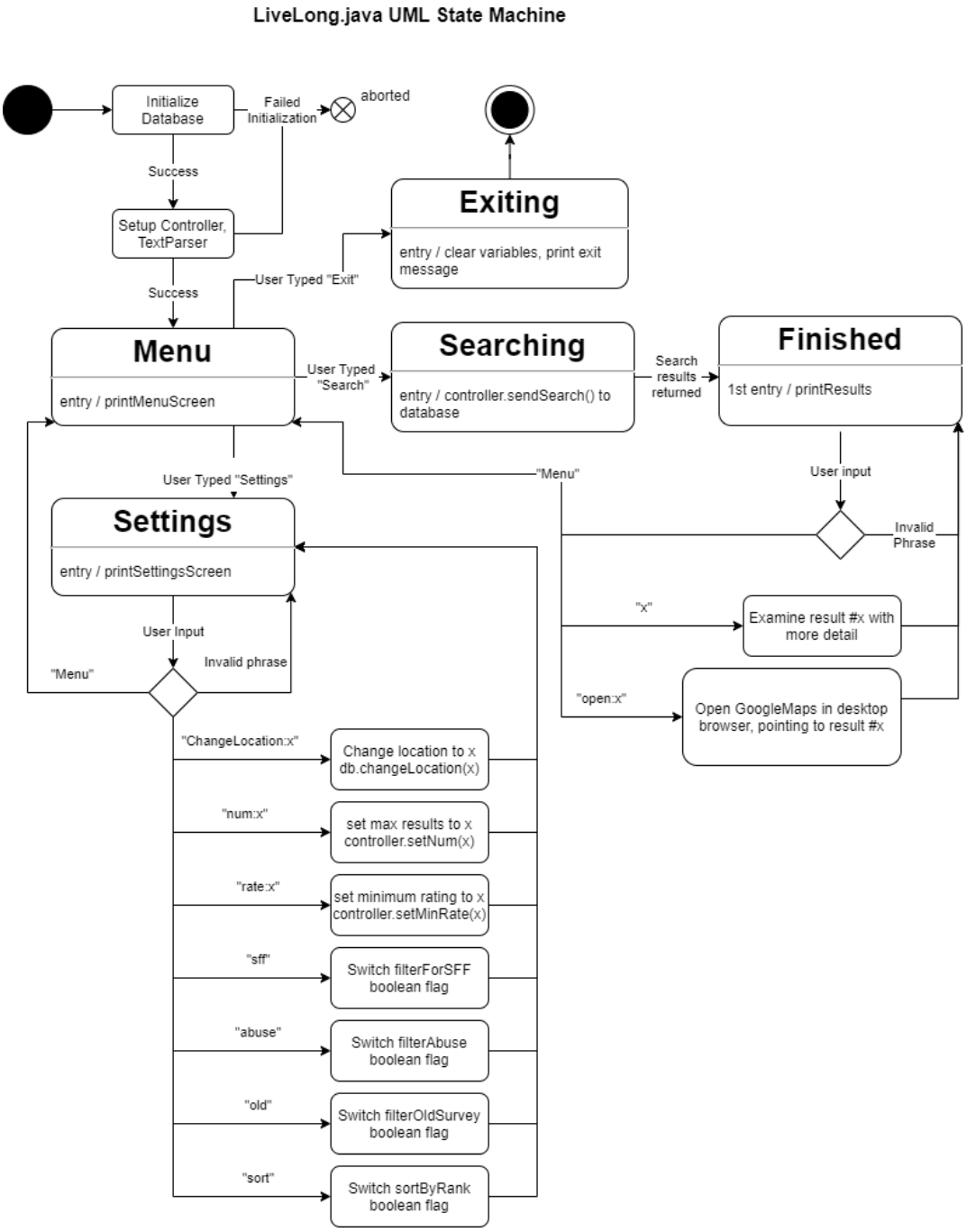


Figure 3: UML State Machine diagram for **LiveLong.java**. This class runs the application loop and interacts with the command line interface to retrieve results from the database. Responses received in the CLI from the user is parsed into appropriate commands, and visual feedback is provided to the user on the current state of the application.

6.4.1 Controller

Has the following private variables

- Database **db** : reference to the **Database** to request information from
- List<NHPair> **results** : results retrieved from **Database.search()**
- int **numResults** : storing longitude value of address
- int **minRating** : storing longitude value of address
- boolean **filterForSFF** : Search Filter - show only NH's that are SFF or is an SFF candidate
- boolean **filterAbuse** : Search Filter - show only NH's that have had no history of abuse
- boolean **filterOldSurvey** : Search Filter - show only NH's that have been inspected recently (< 2 years)
- boolean **sortByRank** : Search Filter - how the **results** should be sorted

When using **sendSearch()**, **Database.search()**, the returned List<NHPair> is stored in **results**. After receiving the results, use **LiveLong.setAppState(State.Finished)** to notify the application that the results have been received.

6.4.2 TextParser

Has the following private methods that each correspond to the given State. The public methods **receive(String, State)** and **printScreen(State)**, depending on the given State, call the corresponding methods.

- void **printScreen(State)** :
  1. State.Menu ⇒ **printMenu()**
  2. State.Settings ⇒ **printSettings()**
  3. State.Finished ⇒ **printFinished()**
  4. State.Exiting ⇒ **printExiting()**
- void **receive(String,State)** :
  1. State.Menu ⇒ **handleMenuResponse(String)**
     - "Search" ⇒ **LiveLong.setAppState(State.Searching)**, prompt **controller.sendSearch()**
     - "Settings" ⇒ **LiveLong.setAppState(State.Settings)**
     - "Exit" ⇒ **LiveLong.setAppState(State.Exiting)**
  2. State.Settings ⇒ **handleSettingsResponse(String)**
     - "ChangeLocation:x" ⇒ **controller.requestChangeLocation(x)**
     - "num:x" ⇒ **controller.setNumResults(x)**
     - "rate:x" ⇒ **controller.setMinRating(x)**
     - "sff" ⇒ Swap Controller.isFilterForSFF
     - "abuse" ⇒ Swap Controller.isFilterAbuse
     - "old" ⇒ Swap Controller.isFilterOldSurvey
     - "sort" ⇒ Swap Controller.isSortByRank
     - "Menu" ⇒ **LiveLong.setAppState(State.Menu)**
  3. State.Finished ⇒ **handleFinished(String)**
     - "x" ⇒ Print out the NursingHome object at index x of **controller.getResults()** with more details
     - "open:x" ⇒ Open in GoogleMaps the user's location and the target NursingHome object at index x of **controller.getResults()**
     - "Menu" ⇒ **LiveLong.setAppState(State.Menu)**

If a state is not mentioned, then there is no method called.

6.5 util package

6.5.1 GoogleMapUtils

A singleton instance, available for any class in application to call. Opens GoogleMaps given a starting and destination address. The given addresses are URL-escaped to be able to be put in the browser URL field. It builds the website/query name with the provided addresses, then prompts the Desktop to open the constructed URL in the default browser.

6.5.2 OpenStreetMapUtils

A singleton instance. Requests geocoding on a string, usually an address. Builds a query to the OpenStreetMap website with the address. A response is received in JSON format. Only the latitude and longitude values are currently saved. Values are saved in a Map, with key "lat" and "lon". If no response is received, returns null.

6.5.3 QuickSort

An implementation of the QuickSort algorithm. Code is directly pulled from the Algorithms 4th Edition textbook used in CS2XB3.

6.5.4 SeparateChainingHashST

An implementation of a Hash symbol table using chaining. Code is directly pulled from the Algorithms 4th Edition textbook used in CS2XB3, and from Lab 9.

6.5.5 SequentialSearchST

An implementation of a symbol table linked list. Code is directly pulled from the Algorithms 4th Edition textbook used in CS2XB3, , and from Lab 9.

7 Internal Review

LiveLong has accomplished its main goal of being able to quickly deliver information about nearby nursing homes to a client. Clients are able to change their location during runtime, and are also able to modify their search filters to get information that is more relevant to their needs. Results also clearly show any warnings that a establishment may have, from ranging to past complaints and penalties, to official warnings and inspections carried out by the government.

With the current iteration, initialization of the application takes less than 10 seconds maximum to parse through four datasets totaling over 500,000 rows of information. Additionally, a compressed version of the datasets is saved locally after the first initialization, allowing for faster load times when launching the app again.

The search functionality also returns results very quickly, making user interaction with the application responsive with little wait times.

The only interaction with the application that results in a longer wait would be changing the user's location, since the application has to request a geocoding process and response from an online service. In this case, that third party service is the only limiting factor in the speed of the application. Even then, the requests usually last three to five seconds maximum, depending on the current traffic load.

In regards to the Requirements Specification document, LiveLong has met most of the requirements. Performance-wise, LiveLong has become a very fast application and delivers results back almost instantaneously. It accurately and reliably returns the expected results each time, and can handle unexpected inputs from the user.

With its good performance and reliability, LiveLong is likely able handle future scalability very well. A doubling in size of the datasets only really affect the initialization times, and not the actual user interactions with the database. The only downside is that newly updated datasets have to be downloaded and manually placed in a location that LiveLong can reach, involving some human interference instead of being a stand-alone application of itself.

Finally, LiveLong was designed with a MVC design pattern and with modularity in mind. This means that it would be simple to implement a new UI interface, or to change the internal implementation of a class (I.e. algorithms, data structures, etc.). Thus, LiveLong can most likely handle future changes with minimal problems.

In short, LiveLong has completed what it set out to do: to deliver important information from official datasets about Nursing Homes to the client in a quick and efficient manner. It is capable of handling many different situations (I.e. changing locations, changing search filters, increased datasets size) and accurately reporting the results back to the client. The speed and efficiency it demonstrates proves that it would be able to handle scalability with larger datasets in the future, and its modular nature allows additions and modifications to its components to be simple and easy to implement. Its weaknesses is mainly in how retrieving updated datasets requires the downloading and movement of a file to a proper folder.