

Get it Done (v2.0)

SFWRENG 2XB3 - Computing and Software - McMaster University

Group 14 - Immanuel Odisho, Ninos Yomo, Paul Heys,
Justin Zhou, Will Donaldson

3 April 2018

The purpose of this document is to provide a description of the classes/modules we have decided to use in our application, and explain why we have decomposed the application into these classes. We have included a UML class diagram showing a static representation of our application classes and the relationship between classes.

Also, for each class, a description of the interface (public entities) as well as a description of the syntax is provided.

Revision Page

Team Members and Roles

Team member	Student No.	Roles/Responsibilities
Immanuel Odisho	400074199	Design Specifications manager Graph processing and GUI researcher
Paul Heys	400069536	Project Leader Sorting Algorithm implementation researcher Sorting algorithm implementer
Ninos Yomo	400062096	ADT developer class relation manager Database researcher
Justin Zhou	400032395	log administrator file manager specifications contributor
Will Donaldson	400072339	Testing Verification and Validation bookkeeper Searching algorithm researcher and implementer

Attestation and Consent:

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 ex-clusive 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.

Contribution Page

Team Members, Roles and Contributions

Team member	Roles/Responsibilities	Contributions
Immanuel Odisho	<i>same as previous page</i>	Design Specifications Document Graph processing GUI, Reviews interface (front end)
Paul Heys	<i>same as previous page</i>	Sorting algorithm Graph Processing (back end)
Ninos Yomo	<i>same as previous page</i>	ContractorADT, GUI splash screen, main menu and search results display (front end)
Justin Zhou	<i>same as previous page</i>	Data Reader module log administrator
Will Donaldson	<i>same as previous page</i>	Searching algorithm verification and validation, Debugging

Executive Summary

The goal of this project is to connect Washingtonians who need contracting work done to the people with the skills to do it. The consumer will be able to enter information about the type of work they want done and how they want it done. This information will be used to identify contractors who meet their needs using the license data of all contractors in the state of Washington. Users will be connected to contractors who specialize in those fields ranked by user given reviews. ¹

¹This abstract was taken from *MileStone1-Group14.docx*

Contents

1	ContractorADT Module	6
1.1	Template Module	6
1.2	Uses	6
1.3	Syntax	6
1.3.1	Exported Types	6
1.3.2	Exported Access Programs	6
1.4	Semantics	6
1.4.1	State Variables	6
1.4.2	State Invariant	7
1.4.3	Assumptions	7
1.4.4	Access Routine Semantics	7
2	Search Module	9
2.1	Template Module	9
2.2	Uses	9
2.3	Syntax	9
2.3.1	Exported Types	9
2.3.2	Exported Access Programs	9
2.4	Semantics	9
2.4.1	State Variables	9
2.4.2	State Invariant	9
2.4.3	Assumptions	9
2.4.4	Access Routine Semantics	9
3	Sort Module	11
3.1	Template Module	11
3.2	Uses	11
3.3	Syntax	11
3.3.1	Exported Types	11
3.3.2	Exported Access Programs	11
3.4	Semantics	11
3.4.1	State Variables	11
3.4.2	State Invariant	11
3.4.3	Assumptions	11
3.4.4	Access Routine Semantics	12
3.4.5	Local Funtions	12

4	Data Reader Module	13
4.1	Template Module	13
4.2	Uses	13
4.3	Syntax	13
4.3.1	Exported Types	13
4.3.2	Exported Access Programs	13
4.4	Semantics	13
4.5	Environment Variables	13
4.5.1	State Variables	13
4.5.2	State Invariant	13
4.5.3	Assumptions	13
4.5.4	Access Routine Semantics	14
5	Reviews Module	15
5.1	Template Module	15
5.2	Uses	15
5.3	Syntax	15
5.3.1	Exported Types	15
5.3.2	Exported Access Programs	15
5.4	Semantics	15
5.4.1	State Variables	15
5.4.2	State Invariant	15
5.4.3	Assumptions	15
5.4.4	Access Routine Semantics	16
6	GUI Package	17
6.1	Package Module	17
6.2	Uses	17
6.3	Syntax	17
6.3.1	Exported Types	17
6.4	Semantics	17
6.5	Environment variables	17
6.5.1	State Variables	17
6.5.2	State Invariant	17
6.5.3	Assumptions	17
6.5.4	Implementation	18
7	UML between public classes	19

1 ContractorADT Module

1.1 Template Module

Contractor

1.2 Uses

N/A

1.3 Syntax

1.3.1 Exported Types

Contractor = ?

1.3.2 Exported Access Programs

Routine name	In	Out	Exceptions
<i>Contractor</i>	<i>String, String, String,String,String, String,String,String,String,\mathbb{Z}</i>	<i>Contractor</i>	
<i>Contractor</i>	<i>String, String, String</i>	<i>Contractor</i>	
isActive		\mathbb{B}	
getLicenseNumber		\mathbb{Z}	
getAddress		<i>String</i>	
getContractorName		<i>String</i>	
getCity		<i>String</i>	
getState		<i>String</i>	
getSpecialty		<i>String</i>	
CompareTo	<i>Contractor</i>	\mathbb{Z}	
avgReview	<i>Map</i>	<i>String</i>	

1.4 Semantics

1.4.1 State Variables

businessName: *String*

licenseNumber: *String*

address: *String*

city: *String*

state: *String*

zip: *String*
 number: *String*
 specialty: *String*
 contractorName: *String*
 activeLicense: \mathbb{Z}

1.4.2 State Invariant

None

1.4.3 Assumptions

The constructor `Contractor` 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.

1.4.4 Access Routine Semantics

`Contractor(Name, License, address, city, state, zip, number, specialty, contractorName, acLicense):`

- transition: *businessName, licenseNumber, address, city, state, zip, number, specialty, contractorName, activeLicense := Name, License, address, city, state, zip, number, specialty, contractorName, acLicense*
- output: *out := self*
- exception: None

`contractor(city1, state1, specialty1):`

- transition: *city, state, specialty := city1, state1, specialty1*
- exception: None

`isActive():`

- output: *out := (activeLicense = 1) \Rightarrow True|False*

`getLicenseNumber():`

- output: *out := licenseNumber*

`getAddress():`

- output: *out := address*

getContractorName():

- output: $out := businessName$

getCity():

- output: $out := city$

getState():

- output: $out := state$

getSpecialty():

- output: $out := specialty$

compareTo(that):

- output: $out := \neg(self.getActive() = that.getActive()) \Rightarrow ((self.getActive() = True) \Rightarrow 1 | False)$

avgReview(map):

- output: $out := Reviews.avgOfContractor(self.getLicenseNumber(), map)$

2 Search Module

2.1 Template Module

Search

2.2 Uses

Contractor
DataReader
Reviews

2.3 Syntax

2.3.1 Exported Types

N/A

2.3.2 Exported Access Programs

Routine name	In	Out	Exceptions
search	seq of Contractor, Contractor, String	seq of Contractor	IOException

2.4 Semantics

2.4.1 State Variables

N/A

2.4.2 State Invariant

None

2.4.3 Assumptions

N/A

2.4.4 Access Routine Semantics

search(Contractors,Contractor,filename):

- output: $\text{out} := \{c : \text{Contractor} \mid c \in \text{Contractors} : ((c.\text{getCity}() = \text{Contractor}.\text{getCity}()) \wedge (c.\text{getState}() = \text{Contractor}.\text{getState}()) \wedge (c.\text{getSpecialty}() = \text{Contractor}.\text{getSpecialty}()) \mid c.\text{getSpecialty}() = \text{General}) \Rightarrow c\}$
- exception: None

3 Sort Module

3.1 Template Module

Sort

3.2 Uses

Contractor
DataReader
Reviews

3.3 Syntax

3.3.1 Exported Types

N/A

3.3.2 Exported Access Programs

Routine name	In	Out	Exceptions
sort	seq of Contractor		
isSorted	seq of Contractor	\mathbb{B}	

3.4 Semantics

3.4.1 State Variables

N/A

3.4.2 State Invariant

None

3.4.3 Assumptions

N/A

3.4.4 Access Routine Semantics

isSorted(Contractors):

- output: $\text{out} := \forall(i : \mathbb{N} | i \in [0..|Contractors|-2] : (Contractors[i].compareTo(Contractors[i+1]) \leq 0))$
- exception: None

sort(Contractors):

- output: $\text{out} := Contractor^n$ such that $\forall(c : Contractor | c \in Contractors : \exists(b : Contractor | b \in B : b.compareTo(c) = 0 \wedge count(c, Contractors) = count(b, B))) \wedge isSorted(B)$
- exception: None

3.4.5 Local Functions

$count(a, A) : Contractor \times Contractor^n$

$count(a, A) \equiv +(i : \mathbb{N} | i \in [0..|A|-1] \wedge A[i].compareTo(a) = 0 : 1)$

4 Data Reader Module

4.1 Template Module

DataReader

4.2 Uses

Contractor

4.3 Syntax

4.3.1 Exported Types

N/A

4.3.2 Exported Access Programs

Routine name	In	Out	Exceptions
readContractors		seq of Contractor	

4.4 Semantics

4.5 Environment Variables

dataset: two dimensional sequence of text characters

4.5.1 State Variables

None

4.5.2 State Invariant

None

4.5.3 Assumptions

None

4.5.4 Access Routine Semantics

readContractors():

- transition: When this method is called it will read through the *FullData.txt* data set and then create a list of Contractor objects and return a list of all the objects made.
- output: out := seq of Contractor
- exception: None

5 Reviews Module

5.1 Template Module

Reviews

5.2 Uses

None

5.3 Syntax

5.3.1 Exported Types

N/A

5.3.2 Exported Access Programs

Routine name	In	Out	Exceptions
initMapFromFile	s: String	Map	
avgOfContractor	licenseNumber:String, Map	string	
addReview	Map, licenseNumber:String, s: String		
writeMapToFile	Map, filename:String		

5.4 Semantics

5.4.1 State Variables

None

5.4.2 State Invariant

None

5.4.3 Assumptions

None

5.4.4 Access Routine Semantics

`initMapFromFile(s):`

- transition: This method is called it will load the *Reviews.txt* database into the program and return a map object of all the Contractors' reviews.
- output: `out := Map` object with contractor license number as key and corresponding contractor's reviews as value.
- exception: None

`avgOfContractor(licenseNumber,Map):`

- output: `out :=` average review of contractor with corresponding license number in the Map object.
- exception: None

`addReview(Map, licenseNumber, s):`

- transition: add the review as a value in Map with the corresponding license number as a key.
- exception: None

`writeMapToFile(Map,filename):`

- transition: write the information in a file with the name of filename only when the program is shutdown.
- exception: None

6 GUI Package

6.1 Package Module

GUI

6.2 Uses

Contractor
Search
Sort
DataReader
Reviews

6.3 Syntax

6.3.1 Exported Types

N/A

6.4 Semantics

6.5 Environment variables

win: two dimensional and interactive sequence of coloured pixels ²

6.5.1 State Variables

None

6.5.2 State Invariant

None

6.5.3 Assumptions

None

²this definition was taken from SFWRENG 2AA4 2018 Assignment 2 specifications.

6.5.4 Implementation

Using the specifications from the other modules, implement the specifications with a graphical user interface in the win environment variable.

7 UML between public classes

