# Group Project, Specification

SFWR ENG 2XB3 - Group 14

April 2, 2018

The purpose of this document is to povide 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.

# Contractor Module

## Template Module

Contractor

## Uses

N/A

## Syntax

### Exported Types

Contractor = ?

### Exported Access Programs

| **Routine name** | **In** | **Out** |
|---|---|---|
| *Contractor* | $String$, $String$, $String$,$String$,$String$,$String$,$String$,$String$,$String$,$\mathbb{Z}$ | *Contracto* |
| *Contractor* | $String$, $String$, $String$ | *Contracto* |
| isActive | | $\mathbb{B}$ |
| getLicenseNumber | | $\mathbb{Z}$ |
| getAddress | | $String$ |
| getContractorName | | $String$ |
| getCity | | $String$ |
| getState | | $String$ |
| getSpecialty | | $String$ |
| CompareTo | $Contractor$ | $\mathbb{Z}$ |
| avgReview | $Map$ | $String$ |

## Semantics

### State Variables

businessName: $String$
licenseNumber: $String$
address: $String$
city: $String$
state: $String$
zip: $String$

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

### State Invariant

None

### 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.

### Access Routine Semantics

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

- transition: $businessName, licenseNumber, address, city, state, zip, number, specialty, contractorNa$
  $Name, License, address, city, state, zip, number, specialty, contractorName, acLicense$
- 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: $out := \neg(self.getActive() = that.getActive()) \Rightarrow ((self.getActive() = True) \Rightarrow 1|False)$

## Search Module

### Template Module

Search

### Uses

Contractor DataReader Reviews

### Syntax

#### Exported Types

N/A

#### Exported Access Programs

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

### Semantics

#### State Variables

N/A

#### State Invariant

None

#### Assumptions

N/A

#### Access Routine Semantics

search(Contractors,Contractor,filename):

- output: out := $\{c : Contractor | c \in Contractors : ((c.getCity() = Contractor.getCity()) \wedge (c.getState() = Contractor.getState()) \wedge (c.getSpecialty() = Contractor.getSpecialty()) | c.getSpecialty() = General) \Rightarrow c\}$

- exception: None

## Sort Module

### Template Module

Sort

### Uses

Contractor DataReader Reviews

### Syntax

#### Exported Types

N/A

#### Exported Access Programs

| **Routine name** | **In** | **Out** | **Exceptions** |
|---|---|---|---|
| sort | seq of Contractor | | |
| isSorted | seq of Contractor | $\mathbb{B}$ | |

### Semantics

#### State Variables

N/A

#### State Invariant

None

#### Assumptions

N/A

#### Access Routine Semantics

isSorted(Contractors):

- output: out $:= \forall(i : \mathbb{N} | i \in [0..|Contractors|-2] : (Contractors[i].compareTo(Contractors[i+1]) <= 0)$

- exception: None

sort(Contractors):

- output: 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

**Local Funtions**

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