JobViz
Design Specifications Document

SFWR 2XB3: Software Engineering Practice and Experience:
Binding Theory to Practice

Software Engineering, McMaster University

April 12, 2020

Group 5

L02

Version 1

Rupinder Nagra, Jonathan Cels, Eshaan Chaudhari, Amir Afzali,
Jarrod Colwell

nagrar5@mcmaster.ca, celsj@mcmaster.ca, chaudhae@mcmaster.ca,
afzalia@mcmaster.ca, colwellj@mcmaster.ca

Revision

Name	Student Number	McMaster Email Address	Role
Rupinder Nagra	400192953	nagrar5@mcmaster.ca	Project Lead
Jonathan Cels	400209614	celsj@mcmaster.ca	Tester, Editor
Eshaan Chaudhari	400193877	chaudhae@mcmaster.ca	Designer
Amir Afzali	400210592	afzalia@mcmaster.ca	Lead Programmer
Jarrod Colwell	400199621	colwellj@mcmaster.ca	Researcher

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.

Contributions

Name	Role	Contributions	Comments
Rupinder Nagra	Project Lead	**Project Proposal** - Prior Work	
Rupinder Nagra	Project Lead	**Project Proposal** - Project Plan	
Rupinder Nagra	Project Lead	**Project Proposal** - Editor	
Rupinder Nagra	Project Lead	**Project Proposal** - References	
Rupinder Nagra	Project Lead	**Proposal Presentation -** Prior Work	
Rupinder Nagra	Project Lead	**Proposal Presentation -** Library/Frameworks	
Rupinder Nagra	Project Lead	**SRS** - Domain	
Rupinder Nagra	Project Lead	**SRS** - Table of Contents	
Rupinder Nagra	Project Lead	**Design Spec.** - Table of Contents	
Rupinder Nagra	Project Lead	**Design Spec.** - Design Review and Evaluation	
Rupinder Nagra	Project Lead	**Meeting Minutes** - Meeting Facilitator	
Rupinder Nagra	Project Lead	**Java Code** - Implemented GraphT	
Rupinder Nagra	Project Lead	**Javadoc** - Added Javadoc to GraphT	

Rupinder Nagra	Project Lead	**Deployment Document -** Uploaded YouTube videos of usage	
Jonathan Cels	Tester, Editor	**Project Proposal** - Abstract	
Jonathan Cels	Tester, Editor	**Proposal Presentation -** Introduction	
Jonathan Cels	Tester, Editor	**Proposal Presentation -** Algorithmic Challenges 1	Section was split into 2 halves
Jonathan Cels	Tester, Editor	**SRS** - Non-Functional Requirements	
Jonathan Cels	Tester, Editor	**SRS** - Editor	
Joanthan Cels	Tester, Editor	**Design Spec. -** Cover Page	
Jonathan Cels	Tester, Editor	**Design Spec. -** Revision Page Setup	
Jonathan Cels	Tester, Editor	**Design Spec. -** Contribution Page Setup	
Jonathan Cels	Tester, Editor	**Design Spec.** - Executive Summary	
Jonathan Cels	Tester, Editor	**Design Spec.** - Design Overview	
Jonathan Cels	Tester, Editor	**Java Code** - Implemented SalaryT	
Jonathan Cels	Tester, Editor	**JUnit Testing -** Implemented TestSalaryT	
Jonathan Cels	Tester, Editor	**JUnit Testing -** Implemented TestSalariesT	

Jonathan Cels	Tester, Editor	**JUnit Testing -** Implemented TestParseT	
Jonathan Cels	Tester, Editor	**JUnit Testing -** Implemented TestPredictionT	
Eshaan Chaudhari	Designer	**Project Proposal** - Algorithmic Challenges	
Eshaan Chaudhari	Designer	**Project Proposal** - Input/Output	
Eshaan Chaudhari	Designer	**Proposal Presentation -** Algorithmic Challenges 2	Section was split into 2 halves
Eshaan Chaudhari	Designer	**Design Spec.** - Created the overall design for the modules for the project	
Eshaan Chaudhari	Designer	**Design Spec.** - Created the Module Interface Specification	
Eshaan Chaudhari	Designer	**Java Code** - Implemented SalariesT module	
Eshaan Chaudhari	Designer	**Java Code** - Implemented SortT module	
Eshaan Chaudhari	Designer	**Java Code** - Implemented merge sort algorithm and comparators for multiple different sorting to be used by the application	
Eshaan Chaudhari	Designer	**Java Code** -	

		Implemented PredictionT module	
Amir Afzali	Lead Programmer	**Project Proposal** - Solution	
Amir Afzali	Lead Programmer	**SRS** - Functional Requirements	
Amir Afzali	Lead Programmer	**SRS** - Development and Maintenance Requirement	
Amir Afzali	Lead Programmer	**Deployment Document -** Created videos explaining usage of application	
Amir Afzali	Lead Programmer	**Java Code** Implemented Application	
Amir Afzali	Lead Programmer	**Java Code** - Implemented ParseT	
Amir Afzali	Lead Programmer	**Java Code** - Implemented AppFrame	
Amir Afzali	Lead Programmer	**Java Code** - Implemented InsightPanel	
Amir Afzali	Lead Programmer	**Java Code** - Implemented InsightOutputFrame	
Amir Afzali	Lead Programmer	**Java Code** - Implemented MenuPanel	
Amir Afzali	Lead Programmer	**Java Code** - Implemented PredictionPanel	
Amir Afzali	Lead Programmer	**Documentation** -	

		Documented AppFrame, InsightOutputFrame, InsightPanel, MenuPanel, PredictionPanel	
Jarrod Colwell	Researcher	**Project Proposal** - Motivation	
Jarrod Colwell	Researcher	**Design Spec. -** UML Revision	
Jarrod Colwell	Researcher	**Javadoc -** Added Javadoc to SalaryT	
Jarrod Colwell	Researcher	**Javadoc -** Added Javadoc to SalariesT	
Jarrod Colwell	Researcher	**Javadoc -** Added Javadoc to SortT	
Jarrod Colwell	Researcher	**Javadoc -** Added Javadoc to PredictionT	
Jarrod Colwell	Researcher	**Javadoc -** Added Javadoc to ParseT	
Jarrod Colwell	Researcher	**Javadoc -** Added Javadoc to Application	
Jarrod Colwell	Researcher	**Design Spec -** Added PredictionT to UML	
Jarrod Colwell	Researcher	**Design Spec -** Added SortT to UML	
Jarrod Colwell	Researcher	**Design Spec -** Added GraphT to UML	

Executive Summary

It is time consuming for current or prospective employees to analyze government data in order to find information regarding the job market. JobViz aims to give users useful and readable data through visualizations and intuitive data querying, in order to streamline job hunting and analysis.

The application takes various inputs selected by the user and partitions the data set such that desired groupings are achieved. The extracted data is used as a basis for querying, numeric prediction, and other aspects of the application's interface.

The algorithmic challenges behind the application include stable sorting, searching, and shortest graphing paths. These operations will be performed as efficiently and quickly as possible.

The dataset used for the project is the Ontario Sunshine List, which includes public-sector salary disclosure on employees paid $100,000 yearly or more. The dataset can be found at https://www.ontario.ca/page/public-sector-salary-disclosure.

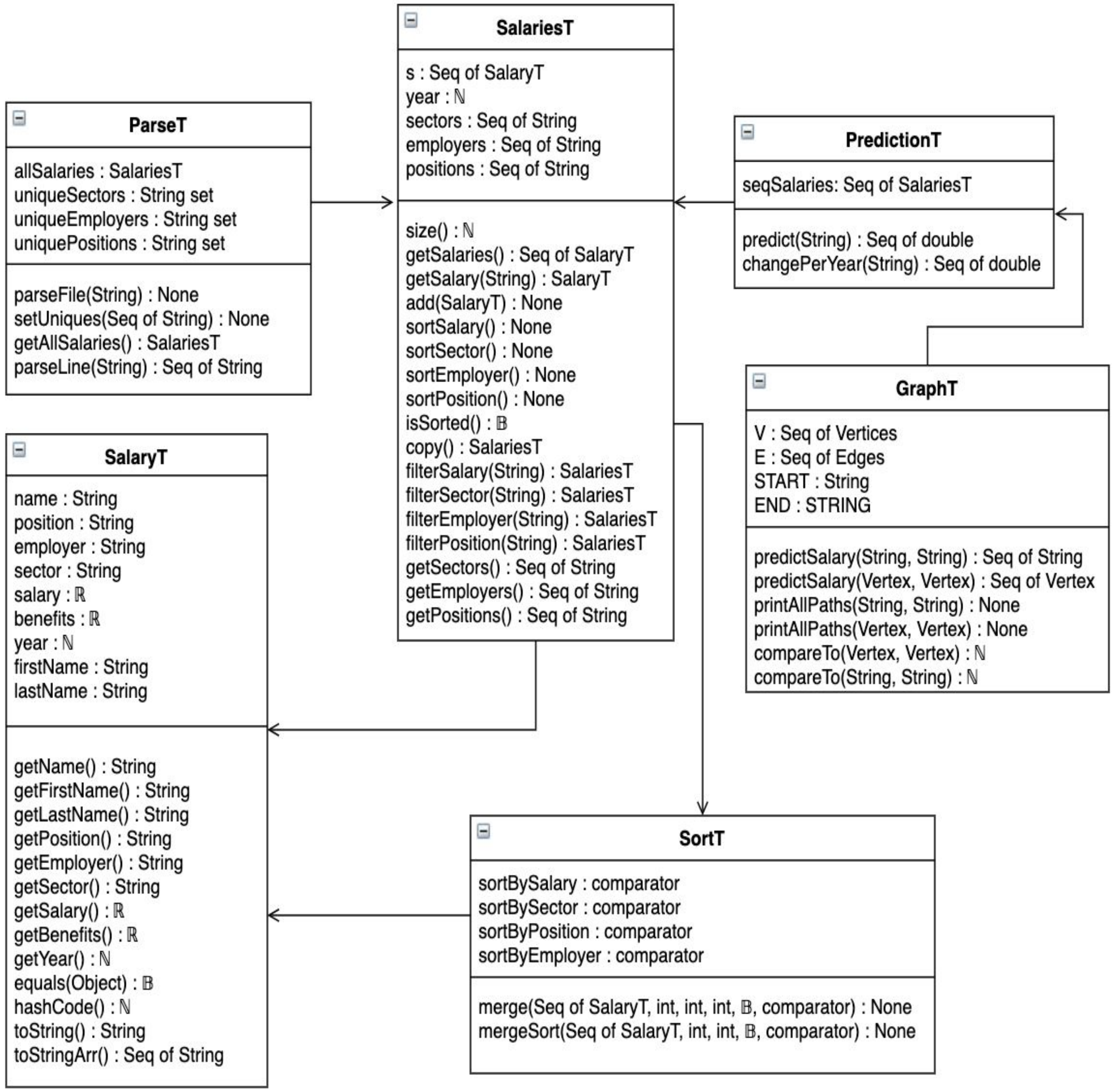
Table of Contents

Design Overview

The Module Interface Specification of the JobViz project is split across six modules: SalaryT, SortT, ParseT, PredictionT, GraphT and SalariesT. SalaryT initializes an object that contains input attributes of columns in the dataset being used; this module represents a single individual's data. The SalariesT module is a dynamic list full of SalaryT objects with access methods such as filtering, adding, and sorting. ParseT is used to parse the CSV dataset files and store them as SalariesT objects, one for each file. PredictionT module is used to give a prediction of the next year's salary for a specific position. It uses SalariesT objects from each year to calculate the prediction. SortT holds the sorting algorithms used to sort different fields from the dataset. ApplicationT runs the program, using GUI to display the data from the other modules and allow for user interactivity. GraphT is the graph implementation that uses Dijkstra's shortest path algorithm used to determine the sector and paths between nodes which will be the average salary difference of a certain occupation.

For the project, many software engineering principles were applied to enhance the overall design and implementation of the product. The design principle of separation of concerns was used to develop distinct sections in which each one addresses its own concern, which is why the modules are decomposed in this manner.

UML


SalariesT
s : Seq of SalaryT
year : ℕ
sectors : Seq of String
employers : Seq of String
positions : Seq of String
size() : ℕ
getSalaries() : Seq of SalaryT
getSalary(String) : SalaryT
add(SalaryT) : None
sortSalary() : None
sortSector() : None
sortEmployer() : None
sortPosition() : None
isSorted() : 𝔹
copy() : SalariesT
filterSalary(String) : SalariesT
filterSector(String) : SalariesT
filterEmployer(String) : SalariesT
filterPosition(String) : SalariesT
getSectors() : Seq of String
getEmployers() : Seq of String
getPositions() : Seq of String
ParseT
allSalaries : SalariesT
uniqueSectors : String set
uniqueEmployers : String set
uniquePositions : String set
parseFile(String) : None
setUniques(Seq of String) : None
getAllSalaries() : SalariesT
parseLine(String) : Seq of String
PredictionT
seqSalaries: Seq of SalariesT
predict(String) : Seq of double
changePerYear(String) : Seq of double
GraphT
V : Seq of Vertices
E : Seq of Edges
START : String
END : STRING
predictSalary(String, String) : Seq of String
predictSalary(Vertex, Vertex) : Seq of Vertex
printAllPaths(String, String) : None
printAllPaths(Vertex, Vertex) : None
compareTo(Vertex, Vertex) : ℕ
compareTo(String, String) : ℕ
SalaryT
name : String
position : String
employer : String
sector : String
salary : ℝ
benefits : ℝ
year : ℕ
firstName : String
lastName : String
getName() : String
getFirstName() : String
getLastName() : String
getPosition() : String
getEmployer() : String
getSector() : String
getSalary() : ℝ
getBenefits() : ℝ
getYear() : ℕ
equals(Object) : 𝔹
hashCode() : ℕ
toString() : String
toStringArr() : Seq of String
SortT
sortBySalary : comparator
sortBySector : comparator
sortByPosition : comparator
sortByEmployer : comparator
merge(Seq of SalaryT, int, int, int, 𝔹, comparator) : None
mergeSort(Seq of SalaryT, int, int, 𝔹, comparator) : None

Module Interface Specification (MIS)

SalaryT: ADT Module

Uses

N/A

Syntax

Exported Access Methods

Method name	In	Out	Exceptions
new SalaryT	String, String, String, $\mathbb{R}$, $\mathbb{R}$, String, String, $\mathbb{N}$	SalaryT	InvalidDataLineException
getName		String	
getFirstName		String	
getLastName		String	
getSalary		$\mathbb{R}$	
getSector		String	
getEmployer		String	
getPosition		String	
getYear		$\mathbb{N}$	
getBenefits		$\mathbb{R}$	
toString		String	
toStringArr		sequence of String	
equals	SalaryT	$\mathbb{B}$	
hashCode		$\mathbb{N}$	

Semantics

State Variables

name: String
firstName: String
lastName: String
salary: $\mathbb{R}$
benefits: $\mathbb{R}$
sector: String
employer: String
position: String
city: String
year: $\mathbb{N}$

Access Method Semantics

new SalaryT(sector, firstName, lastName, salary, benefits, employer, position, year):

- transition: name, firstName, lastName, salary, benefits, sector, employer, position, year := firstName + “ “ + lastName, firstName, lastName, salary, benefits, sector, employer, position, year
- output: out := SalaryT
- exception: (salary < 100000 ⇒ InvalidDataLineException) | (benefits < 0 ⇒ InvalidDataLineException) | (year < 0 ⇒ InvalidDataLineException)

getName():

- output: out := name
- exception: none

getFirstName():

- output: out := firstName
- exception: none

getLastName():

- output: out := lastName
- exception: none

getSalary():

- output: out := salary
- exception: none

getBenefits():

- output: out := benefits
- exception: none

getSector ():

- output: out := sector
- exception: none

getEmployer():

- output: out := employer
- exception: none

getPosition():

- output: out := position
- exception: none

getYear():

- output: out := year
- exception: none

toString():

- output: out := sector + " " + employer + " " + position + " " + salary
- exception: none

toStringArray():

- output: out := (sector, employer, position, salary)
- exception: none

equals(that):

- output: out := (that = null ⇒ false) | ((that.sector = sector) ^ (that.employer = employer) ^ (that.position = position) ^ (that.name = name) ^ (that.salary = salary) ^ (that.year = year) ⇒ true)
- exception: none

hashCode():

- output: out := (∀ s, t | s, t : SalaryT ⇒ s.hashCode ≠ t.hashCode)
- exception: none

Assumptions

No methods are called before a SalaryT object has been initialized. Inputs are assumed to be of the correct format and type.

Requirements

SalaryT allows every input to be checked for validity when it is first generated from the dataset. It throws exceptions in the case of abnormal circumstances, fulfilling the requirement that input is checked for validity.

SortT: Template Module

Uses
N/A

Syntax
Exported Access Methods

Method name	In	Out	Exceptions
new SortT		SalariesT	
mergeSort	Sequence of SalaryT, ℕ, ℕ, 𝔹, Comparator		

Semantics
State Variables
None

Access Method Semantics

mergeSort(array, l, h, ascending, c):
- output: transition := sort array using merge sort by using the comparator c and boolean value that represents if array is to be sorted ascending or descending
- exception: none

Additional Information
This class also contains many static classes that implement the comparator interface in order to be used in the mergeSort method to sort the array in different ways.

SalariesT: Template Module Inherits SortT

Uses

SalaryT, SortT

Syntax

Exported Access Methods

Method name	In	Out	Exceptions
new SalariesT	Sequence of SalaryT, year	SalariesT	
getSalaries		Array of SalaryT	
getSalary	$\mathbb{Z}$	SalaryT	
add	SalaryT		
isSorted		$\mathbb{B}$	
newHash	Function	HashT	
filterSalary	$\mathbb{Z}$, $\mathbb{Z}$	SalariesT	
filterSector	String	SalariesT	
filterPosition	String	SalariesT	
filterEmployer	String	SalariesT	
sortSalary	String		
sortPosition	$\mathbb{B}$		
sortSector	$\mathbb{B}$		
sortEmployer	$\mathbb{B}$		
positionMean	String	$\mathbb{R}$	

Semantics
State Variables
s: sequence of SalaryT
year: ℕ

Access Method Semantics
new SalariesT(salaries, year):
- transition: s, year := salaries, year
- output: out := SalariesT
- exception: none

getSalaries():
- output: out := s
- exception: none

getSalary(name):
- output: out := find SalaryT in s matching name
- exception: none

size():
- output: out := size of array s
- exception: none

add(e):
- transition: s := add e to salaries s
- exception: none

isSorted():
- output: out := is s sorted?
- exception: none

filterSalary(low, high):
- output: out := filter s and return new SalariesT object containing salaries that are within the low and high values
- exception: exc:= throw rangeException if low > high

filterSector(n):
- output: out := filter s and return new SalariesT object containing salaries that are from the sector s
- exception: none

filterSector(n):
- output: out := filter s and return new SalariesT object containing salaries that are from the sector s

- exception: none

filterEmployers(n):
- output: out := filter s and return SalariesT object containing salaries that are from the same employer n
- exception: none

sortSalary(b):
- output: out := mergeSort(s, 0, size() - 1, b, comparator for sorting by salary)
- exception: none

sortPosition(b):
- output: out := mergeSort(s, 0, size() - 1, b, comparator for sorting by position)
- exception: none

sortEmployer(b):
- output: out := mergeSort(s, 0, size() - 1, b, comparator for sorting by employer)
- exception: none

sortSector(b):
- output: out := mergeSort(s, 0, size() - 1, b, comparator for sorting by sector)
- exception: none

positionMean(p):
- output: out := the mean of salaries in s whose position is p
- exception: exc := throw rangeException if 1 year or less of data

Requirements

This module allows the program to output a list of salaries, fulfilling one of the two output requirements. This module also allows for the utility requirement, as this data structure allows for efficient sorting and searching algorithms to be used on the SalaryT objects by inheriting the SortT module.

PredictionT: Template Module

Uses

SalaryT, SalariesT

Syntax

Exported Access Methods

Method name	In	Out	Exceptions
new PredictionT	Sequence of SalariesT	PredictionT	
predict		Sequence of $\mathbb{R}$	

Semantics

State Variables

s: sequence of SalariesT

Access Method Semantics

new PredictionT(salaries):

- transition: s := salaries
- output: out := PredictionT
- exception: none

predict(p):

- output: out := find the mean of salaries for all SalariesT objects for different years in s for position p, find the percentage of change between the mean salary for consecutive years, find the difference of the percentage values for the last two years to determine the percentage change for next year, use it to predict salary for next year for position p, return an array where first value is prediction value and the second value is the percentage of change from previous year
- exception: returns null if not enough data in s for prediction of salary for position p

Requirements:

This module allows the program to output a future salary prediction for a given position, fulfilling one of the two output requirements.

ParseT: Template Module

Uses

SalaryT, SalariesT

Syntax

Exported Access Methods

Method name	In	Out	Exceptions
new ParseT	String	ParseT	
parseFile	String		fileNotFoundException
getAllSalaries		SalariesT	

Semantics

State Variables

allSalaries: SalariesT
uniqueSectors: Hashset<String>
uniquePositions: Hashset<String>
uniqueEmployers: Hashset<String>

Access Method Semantics

new ParseT(e):

- transition: allSalaries, uniqueSectors, uniquePositions, uniqueEmployers := parse(e)
- output: out := ParseT
- exception: none

parse():

- transition: allSalaries, uniqueSectors, uniquePositions, uniqueEmployers := read file and create SalaryT objects for SalariesT object, list of unique sectors, list of unique positions, list of unique employers
- exception: none

getAllSalaries():

- output: out := allSalaries

- exception: none

Requirements

This module allows for input to be taken from the data files. This fulfills the requirement of taking input in the form of data requests. The module also formats the data for input into the SalaryT module, allowing the SalaryT module to check for invalid inputs.

GraphT: Template Module

Uses
SalaryT, SalariesT

Syntax
Exported Access Methods

Method name	In	Out	Exceptions
new GraphT	SalariesT	GraphT	
predictSalary	Vertex, Vertex	Sequence of Vertex	
predictSalary	String, String	Sequence of String	
compareTo	Vertex, Vertex	$\mathbb{N}$	
compareTo	String, String	$\mathbb{N}$	
printAllPaths	String		

Semantics
State Variables
graph : sequence of (string, vertex) tuples

Access Method Semantics
new GraphT(s):
- transition: g := an adjacency matrix of integers that represents a graph where each node is a SalaryT
- exception: none

predictSalary(s1: Vertex, s2: Vertex):
- output: out := closest path from node s1 to node s2
- exception: none

predictSalary(s1: String, s2: String):

- output: out := closest path from node s1 to node s2
- exception: none

compareTo(s1: Vertex, s2: Vertex):
- output: out := compares node s1 to node s2
- exception: none

compareTo(s1: String, s2: String):
- output: out := compares node s1 to node s2
- exception: none

printAllPaths(s1: String):
- output: out := prints all the paths from dijkstra’s algorithm
- exception: none

Design Review and Evaluation

This critique will self-assess several software qualities to the MIS shown above. The quality of consistency looks for appropriate naming conventions, the ordering of parameters in argument lists, and exception handling. This is a quality that is achieved based on the modules above. The naming conventions in the document are consistent and do not differ from previously defined variables or methods. The same variable names are used in the access routine semantics and the exported access program sections, where there is no discrepancy between the two. The variable names and methods are easy to read and understand, and have semantic meaning to them for the reader to understand. The function parameters are also consistent with each other, and the order does not change throughout the document.

The specification omits unnecessary features in it, also fulfilling the quality of essentiality. Every redundant access method is removed, and every other method is of use to the specification. For example, the SalariesT access programs include deleting, restoring, and filtering the position of a certain SalaryT object. This makes the module very simplistic and essential as no other method or combination of methods can be used to perform another function, fulfilling the quality.

The program is also a general one, as it is very implementation independent. The implementation is not shown in the MIS, as the MIS has been solely designed to be able to be efficiently understood semantically and syntactically, and no explanation on the implementation is given, as this would take away from this property. Only the MIS language has been used.

It is also minimal, as it avoids containing access routines that can be separated into two potentially independent services. For example, the access programs in the SalaryT module all serve their own individual purpose, and cannot be separated in the module. This is because they all serve different purposes so one cannot be used for another method.

The MIS above also shows high cohesion within the modules, as all the components within them are very closely related to one another. For example, in the SalariesT module, calling the sortSector method causes a call to another method within the module, also calling other possible outputs of other access methods in it, such as getSalaries. This is an example of high cohesion in one of the modules, as the components are closely connected.

There are certain parts in the implementation of the MIS that are hidden from clients, which is the fundamental concept behind information hiding. The use of private/protected variables and methods and how the modules are inherited in the Java files, ensure that the client can only view only anything relevant to them, while the rest is hidden. The principle of encapsulation was also used in the Java implementation of the MIS above, which is heavily related to the concept of information hiding. Therefore, this critique has demonstrated the inclusion of all important software qualities expressed above.

UML STATE DIAGRAM (SortT)