

# Design Specification

**Version: 1**

**Project Name:** Earthquake Risk Assessment

**Group:** 01

**Member:** Kan Hailan, Sembakutti Kalindu,  
Tao Haoyang, Ye Fang

April 11, 2020

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.

## Revision History

| Name                 | Date           | Version |
|----------------------|----------------|---------|
| Design Specification | 11 April, 2020 | 1       |
|                      |                |         |
|                      |                |         |

## Group Members and Roles

| Name               | Student Number | Role in the Project         | Responsibility  |
|--------------------|----------------|-----------------------------|---|
| Kan Hailan         | 400207974      | client & tester             | Implement unit tests, debug and verify code   |
| Sembakutti Kalindu | 1046206        | researcher & programmer     | Cleaning and extracting data, writing documents, data structure algorithms                              |
| Tao Haoyang        | 400171589      | designer & programmer       | Module decomposition, implement ADT modules.  |
| Ye Fang            | 400273067      | project leader & programmer | Manage the project, produce the prototype, implement sort, risk assessment, controller and view modules |

## Contribution

| Name               | Role(s)                     | Contributions   |
|--------------------|-----------------------------|---|
| Kan Hailan         | client & tester             | <ul style="list-style-type: none"> <li>• Write test cases for 12 modules</li> <li>• Debug or change test cases according to test results</li> <li>• Review and verify MIS</li> <li>• Finish the domain part in SRS</li> <li>• Finish the trace back to requirements and internal review in Design Specification</li> </ul>  |
| Sembakutti Kalindu | researcher & programmer     | <ul style="list-style-type: none"> <li>• Outlining initial SRS functional and non-functional requirements</li> <li>• Implementing CSVreader module</li> <li>• Implementing data structures (EarthquakeBag, RedBlackBST, Queue)</li> <li>• Modified datasets to have correct province names, and non-empty cells</li> <li>• The MIS of 4 modules (CSVreader, RedBlackBST, Earthquake T, Queue)</li> <li>• Review and editing documents</li> </ul>  |
| Tao Haoyang        | designer & programmer       | <ul style="list-style-type: none"> <li>• Create UML class diagram for all initial 21 modules and keep updating to current version.</li> <li>• Idea discussions with project leader.</li> <li>• Implementation of ADT modules (EarthquakeT, CityT, PointT)</li> <li>• Implementation of GeoCollection module</li> <li>• SRS part 3.4: Requirements on the development and maintenance process</li> <li>• Design Specification part 2: Module decomposition and UML, view of uses relationship</li> </ul>                                     |
| Ye Fang            | project leader & programmer | <ul style="list-style-type: none"> <li>• Manage the program to meet all milestones</li> <li>• Finish the MIS of 14 modules</li> <li>• Implement 12 modules (CityPostT, Edge, Graph, SearchEarthquakes, Sort, RiskAssessement, ViewRisk, ViewList, DisplayByMagnitude, DisplayByDistance, Controller, MCVDemo)</li> <li>• Prepare the outline of SRS and Design Specification</li> <li>• Finish and update the Overall Description in SRS</li> <li>• Design Specification part 3: MIS (14 modules) and UML state machine diagrams</li> </ul> |

## **Executive Summary**

The earthquake risk assessment is vital for numbers of different reasons. The main reason is that human lives are endangered unnecessarily by living in a geologically active area that poses a significant risk of a damaging earthquake in the future. There are places considered to be geographically beautiful, but they have an underlying earthquake risk. First, the people living in these places should become aware of this risk, in order to, make informed decisions for future relocations.

The other aspect is that cities should adjust their building codes to be more suitable to the geological activities. The cities can pass new construction codes into by-laws, based on our earthquake risk assessment, so the building developers who are more profit-oriented do not endanger lives unnecessarily by over-doing construction project in a geologically active place.

The last aspect is insuring the buildings that were developed in the past, but now face a significant risk due to known geological activities. These buildings can be Canadian heritage sites that may require significant costs to repair if damaged by an earthquake. With the knowledge of our risk assessment in hand, all stakeholders can take necessary steps to properly insure these buildings.

# Table of Contents

|   |           |
|---|-----------|
| <b>1 Overall Design Description .....</b>     | <b>5</b>  |
| <b>2 Module Decomposition .....</b>           | <b>5</b>  |
| <b>3 MIS .....</b>                            | <b>6</b>  |
| 3.1 MIS .....                                 | 6         |
| 3.2 state machine diagrams .....              | 44        |
| <b>4 Internal Review and Evaluation .....</b> | <b>45</b> |

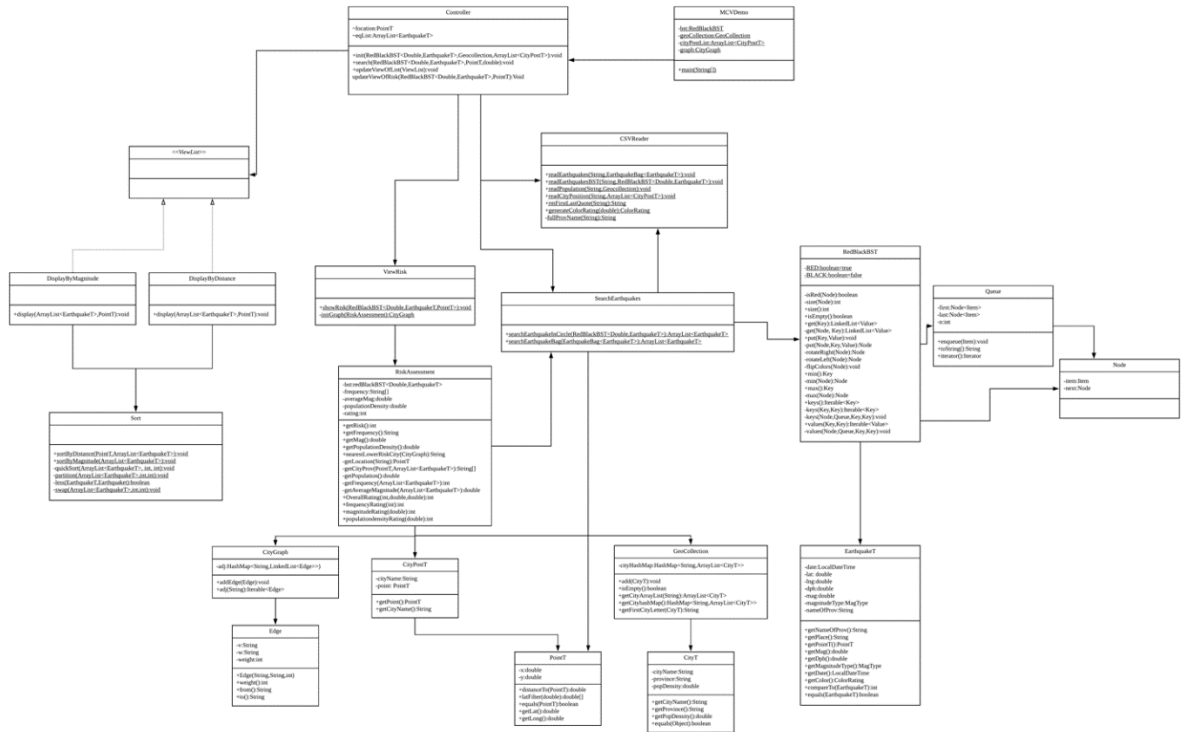
# 1 Overall Design Description

The project reads from a csv file all the earthquake information, one earthquake at a time, storing individual earthquake information in an earthquake object, and those objects themselves in a RedBlackBST data structure. The project allows the user to provide input latitude, longitude, and a radius to search earthquakes within the radius. It produces a list of earthquakes sorted based on the magnitude or distance from the user location. It also shows the risk rating for a specific location, based on earthquakes and population data within a 100 km radius. It calculates the overall risk rating, based on earthquake frequency, average magnitude, and population density.

In addition, the project uses a city coordinate dataset which has a specific location of a city in terms of longitude and latitude. The program iterates through all the earthquakes within a 100 km radius, finds the closest earthquake, and accesses its location in order to determine a geographical name of the current location. It, then, use a CityGraph created by the city coordinate dataset to determine the closest city by finding the adjacent edge with the lowest weight from the current location. When iterating through the adjacent cities, it also determines if the closest city also has a lower risk rating than the current location. Once, it finds the closest city with a lower risk rating, it outputs that location.

## 2 Module Decomposition

- Our product is following the MVC design pattern that is, we have a controller module and view modules. At the very beginning of our prototype, the performance was unsatisfiable, since we are using EarthquakeBag, a linked list data structure to keep everything in order.
- Then, we decide to add a method “latFilter” in PointT ADT. This method will be given a radius and make the current PointT object as the center and produce a new pair of latitude to “filter” out points that outside of the given radius. Now the performance has increased a little.
- We finally decide to use RedBlackBST to improve the performance, discard using EarthquakeBag. By making this decision, the time complexity for insertion has increased, but when it comes to searching and sorting, the time taken is greatly reduced.
- All classes and ADTs are well reused. For example, PointT is a point consist of x and y coordinates representing latitude and longitude respectively, and CityPostT uses PointT as part of its state variable. By dividing those classes and methods, we are trying to balance low coupling, code reusability, and maintainability at the same time.
- UML class [diagram](#)(←click on the hyperlink to view in full resolution)



# 3 MIS

There are totally 19 modules for this system. We have the MISEs for all modules except MCVDemo in the following section. We also include the state machine diagrams for Sort and MCVDemo classes respectively.

## 3.1 MIS

# CSVreader Module

## Module

CSVreader

## Uses

CityPostT, CityT, EarthquakeT,  
EarthquakeT.ColorRating, EarthquakeT.MagType,  
EarthquakeBag, GeoCollection, RedBlackBST

## Syntax

### Exported Constants

None

### Exported Access Programs

| Routine name        | In                       | Out         | Exceptions  |
|---------------------|--------------------------|-------------|-------------|
| readEarthquakes     | String, EarthquakeBag    |             | IOException |
| readEarthquakesBST  | String, RedBlackBST      |             | IOException |
| readPopulation      | String, GeoCollection    |             | IOException |
| readCityPosition    | String, seq of CityPostT |             | IOException |
| rmFirstLastQuote    | String                   | String      |             |
| generateColorRating | $\mathbb{R}$             | ColorRating |             |
| fullProvName        | String                   | String      |             |

## Semantics

### Environment Variables

A file listing Earthquake Information.  
A file listing Population Densities.  
A file listing City Coordinates.

### State Variables

None



## State Invariant

None

## Assumptions

The input file by filename will match the specific format required by the read method.

## Access Routine Semantics

readEarthquakes(filename, bag):

- transition: Read each line of the earthquake csv file and convert to a EarthquakeT object, which is stored in a EarthquakeBag.
- exception: If the file by filename does not exist, produces IOException.

readEarthquakesBST(filename, bst):

- transition: Read each line of the earthquake csv file and convert to a EarthquakeT object, which is stored in a RedBlackBST.
- exception: If the file by filename does not exist, produces IOException.

readPopulation(filename, geoCollec):

- transition: Read each line of the population csv file and convert to a CityT object, which is stored in a GeoCollection HashMap.
- exception: If the file by filename does not exist, produces IOException.

readCityPosition(filename, cityPostList):

- transition: Read each line of the city coordinates csv file and convert to a CityPostT object, which is stored in a list of cities.
- exception: If the file by filename does not exist, produces IOException.

rmFirstLastQuote(cell):

- transition: Remove first and last double quotations from a string value.
- exception: None

generateColorRating(cell4):

- transition: Generate an enum ColorRating type based on the magnitude of earthquake.
- exception: None

fullProvName(nameP):

- output: a new province name similar to the following table.

|            | nameP =          | <i>out</i> :=             |
|------------|------------------|---------------------------|
| nameP  = 2 | ON               | Ontario                   |
|            | QC, PQ           | Quebec                    |
|            | NS               | Nova Scotia               |
|            | NB               | New Brunswick             |
|            | MB               | Manitoba                  |
|            | BC               | British Columbia          |
|            | PE               | Prince Edward Island      |
|            | SK               | Saskatchewan              |
|            | AB               | Alberta                   |
|            | NL               | Newfoundland and Labrador |
|            | NU               | Nunavut                   |
|            | NT               | Northwest Territories     |
|            | YT               | Yukon                     |
|            | AK               | Alaska                    |
|            | WA               | Washington                |
|            | default          | UNLOCATED                 |
| nameP  ≠ 2 | VANCOUVER ISLAND | British Columbia          |
|            | SOUTHERN QUEBEC  | Quebec                    |
|            | default          | UNLOCATED                 |

- exception: None

## Considerations

There are a number of different variations of geolocation names in the earthquake csv file, for these an appropriate province name should be assigned. For any that could not be matched to a province name, UNLOCATED should be assigned.

# Point ADT Module

## Template Module

PointT

## Uses

N/A

## Syntax

### Exported Types

PointT = ?

### Exported Access Programs

| Routine name | In                       | Out                          | Exceptions       |
|--------------|--------------------------|------------------------------|------------------|
| new PointT   | $\mathbb{R}, \mathbb{R}$ | PointT                       | RuntimeException |
| getLat       |                          | $\mathbb{R}$                 |                  |
| getLong      |                          | $\mathbb{R}$                 |                  |
| distance     | PointT                   | $\mathbb{R}$                 |                  |
| latFilter    | $\mathbb{R}$             | $< \mathbb{R}, \mathbb{R} >$ |                  |
| equals       | PointT                   | $\mathbb{B}$                 |                  |

## Semantics

### State Variables

$x : \mathbb{R}$

$y : \mathbb{R}$

### Access Routine Semantics

PointT( $lat, long$ ):

- transition:  $x, y := lat, long$
- output:  $out := self$
- exception:  $exc := ((lat > 90 \vee lat < -90) \Rightarrow \text{IndexOutOfBoundsException})$

getLat():

- output:  $out := x$

getLong():

- output:  $out := y$

distanceTo(that):

- output:  $out := d$  such that  $d$  is the distance(in km) between current point and that

latFilter(radius):

- output:  $out := \langle minLat, maxLat \rangle$  such that  $\forall (p : PointT | distanceTo(p) \leq radius : p.getLat() \geq minLat \wedge p.getLat() \leq maxLat$

equals(that):

- output:  $out := (x = that.getLat()) \wedge (y = that.Long())$

# City ADT Module

## Template Module

CityT

## Uses

N/A

## Syntax

### Exported Types

CityT = ?

### Exported Access Programs

| Routine name  | In                           | Out          | Exceptions |
|---------------|------------------------------|--------------|------------|
| new CityT     | String, String, $\mathbb{R}$ | CityT        |            |
| getCityName   |                              | String       |            |
| getProvince   |                              | String       |            |
| getPopDensity |                              | $\mathbb{R}$ |            |
| equals        | CityT                        | $\mathbb{B}$ |            |

## Semantics

### State Variables

$cityName : \text{String}$   
 $province : \text{String}$   
 $popDensity : \mathbb{R}$

### Access Routine Semantics

$\text{CityT}(city, pro, pop):$

- transition:  $cityName, province, popDensity := city, pro, pop$
- output:  $out := self$

$\text{getCityName}():$

- output:  $out := cityName$

$\text{getProvince}():$

- output:  $out := province$

$\text{getPopDensity}():$

- output:  $out := popDensity$

$\text{equals}(that):$

- output:  $out := (cityName = that.getCityName()) \wedge (province = that.getProvince()) \wedge (popDensity = that.getPopDensity())$

# City Position ADT Module

## Template Module

CityPostT

## Uses

PointT

## Syntax

### Exported Types

CityPostT = ?

### Exported Access Programs

| Routine name  | In                                  | Out       | Exceptions |
|---------------|-------------------------------------|-----------|------------|
| new CityPostT | String, $\mathbb{R}$ , $\mathbb{R}$ | CityPostT |            |
| getPoint      |                                     | PointT    |            |
| getCityName   |                                     | String    |            |

## Semantics

### State Variables

*cityName* : String

*point* : PointT

### Access Routine Semantics

CityT(*city*, *lat*, *lon*):

- transition: *cityName*, *point* := *city*, *newPointT*(*lat*, *lon*)
- output: *out* := *self*

getPoint():

- output: *out* := *point*

getCityName():

- output: *out* := *cityName*

# CityT Collection Module

## Template Module

GeoCollection

## Uses

CityT

## Syntax

### Exported Types

GeoCollection = ?

### Exported Access Programs

| Routine name | In     | Out                                   | Exceptions |
|--------------|--------|---------------------------------------|------------|
| add          | CityT  |                                       |            |
| getCities    | String | seq of CityT                          |            |
| getAllCities |        | set of tuple of(String, seq of CityT) |            |
| isEmpty      |        | $\mathbb{B}$                          |            |



## Semantics

### State Variables

$s$  : set of tuple of (String, seq of CityT)

### Access Routine Semantics

add(city):

- transition:  $s := \{ \langle str, cities \rangle : \langle String, \text{seq of CityT} \rangle \mid \langle str, cities \rangle \in s : (str = \text{getFirstCityLetter}(city) \Rightarrow \langle str, cities || [city] \rangle \mid true \Rightarrow \langle str, cities \rangle) \}$

isEmpty():

- output:  $out := (|s| = 0 \Rightarrow true \mid true \Rightarrow false)$

getCities(firstLetter):

- output:  $out := \{ \langle str, cities \rangle : \langle String, \text{seq of CityT} \rangle \mid \langle str, cities \rangle \in s \wedge str = \text{firstLetter} : cites \}$

getAllCities():

- output:  $out := s$

### Local Functions

getFirstCityLetter : String  $\rightarrow$  String

getFirstCityLetter(city)  $\equiv$  city[0]

# Earthquake ADT Module

## Template Module

EarthquakeT

## Uses

LocalDateTime, PointT

## Syntax

### Exported Types

EarthquakeT = ?

ColorRating = { NOCOLOR, ZERO, PURPLE, BLUE, GREEN, YELLOW, ORANGE, RED }

MagType = { M5, mb, MB, Mb, MC, Mc, mc, ML, MLSn, MN, MS, MW, Ms, Mw, BLANK }

*# EarthquakeT implements Comparable(EarthquakeT)*

### Exported Constants

None

## Exported Access Programs

| Routine name    | In   | Out           | Exceptions |
|-----------------|--|---------------|------------|
| new EarthquakeT | String, String, LocalDateTime,<br>$\mathbb{R}$ , $\mathbb{R}$ , $\mathbb{R}$ , $\mathbb{R}$ , MagType, ColorRating | EarthquakeT   |            |
| getNameOfProv   |  | String        |            |
| getPlace        |  | String        |            |
| getPointT       |  | PointT        |            |
| getMag          |  | $\mathbb{R}$  |            |
| getDph          |  | $\mathbb{R}$  |            |
| getMagitudeType |  | MagType       |            |
| getDate         |  | LocalDateTime |            |
| getColor        |  | ColorRating   |            |
| compareTo       | EarthquakeT  | $\mathbb{Z}$  |            |
| equals          | EarthquakeT  | $\mathbb{B}$  |            |

## Semantics

### State Variables

place: String  
nameOfProv: String  
date: LocalDateTime  
lat:  $\mathbb{R}$   
lng:  $\mathbb{R}$   
dph:  $\mathbb{R}$   
mag:  $\mathbb{R}$   
magnitudeType: MagType  
color: ColorRating

### State Invariant

None

### Assumptions

Two earthquakes are not the same if they happened to have two different dates or two different places recorded.

## Access Routine Semantics

EarthquakeT(place, prov, date, lat, lng, dph, mag, mgT, color):

- transition:  
lat, lng, place, nameOfProv, date, dph, mag, magnitudeType, color :=  
lat, lng, place, prov, date, dph, mag, mgT, color
- output: *out* := *self*
- exception: None

getNameOfProv():

- output: *out* := nameOfProv
- exception: None

getPlace():

- output: *out* := place
- exception: None

getPointT():

- output: *out* := PointT(lat, lng)
- exception: None

getMag():

- output: *out* := mag
- exception: None

getDph():

- output: *out* := dph
- exception: None

getMagitudeType():

- output: *out* := magnitudeType
- exception: None

getDate():

- output:  $out := \text{date}$
- exception: None

getColor():

- output:  $out := \text{color}$
- exception: None

compareTo(eq):

- output:  $out :=$  an integer value according to the following table.

|                     | $out :=$ |
|---------------------|----------|
| $this.mag < eq.mag$ | -1       |
| $this.mag > eq.mag$ | 1        |
| $this.mag = eq.mag$ | 0        |

- exception: None

equals(that):

- output:  $out := (\text{sameDate} \wedge \text{samePoint} \wedge \text{samePlace} \wedge \text{sameDepth} \wedge \text{sameMagValue} \wedge \text{sameMagType} \wedge \text{sameEqClass}) \Rightarrow \text{True} | \text{True} \Rightarrow \text{False}$
- exception: None

## Local Functions

sameDate: EarthquakeT  $\rightarrow$   $\mathbb{B}$

sameDate( $d$ )  $\equiv (d.date) = (this.date)$

*# Returns true if the given EarthquakeT object has the same date as the current.*

samePoint: EarthquakeT  $\rightarrow$   $\mathbb{B}$

samePoint( $d$ )  $\equiv (d.Point) = (this.Point)$

*#Returns true if the given EarthquakeT object has the same Point as the current.*

samePlace: EarthquakeT  $\rightarrow$   $\mathbb{B}$

samePlace( $d$ )  $\equiv (d.place) = (this.place)$

*#Returns true if the given EarthquakeT object has the same place as the current.*

sameDepth: EarthquakeT  $\rightarrow$   $\mathbb{B}$

sameDepth( $d$ )  $\equiv |d.dph - this.dph| < 0.0000001$

*#Returns true if the given EarthquakeT object has the same depth value as the current within the tolerance.*

sameMagValue: EarthquakeT  $\rightarrow$   $\mathbb{B}$

sameMagValue( $d$ )  $\equiv |d.mag - this.mag| < 0.0000001$

*#Returns true if the given EarthquakeT object has the same magnitude value as the current within the tolerance.*

sameMagType: EarthquakeT  $\rightarrow$   $\mathbb{B}$

sameMagType( $d$ )  $\equiv (d.magnitudeType) = (this.magnitudeType)$

*#Returns true if the given EarthquakeT object has the same magnitude type as the current.*

sameEqClass: EarthquakeT  $\rightarrow$   $\mathbb{B}$

sameEqClass( $d$ )  $\equiv (d.color) = (this.color)$

*#Returns true if the given EarthquakeT object has the same earthquake class as the current.*

## Edge ADT Module

### Template Module

Edge

### Uses

N/A

### Syntax

#### Exported Types

Edge = ?

#### Exported Access Programs

| Routine name | In                           | Out          | Exceptions |
|--------------|------------------------------|--------------|------------|
| new Edge     | String, String, $\mathbb{Z}$ | Edge         |            |
| weight       |                              | $\mathbb{Z}$ |            |
| from         |                              | String       |            |
| to           |                              | String       |            |

## Semantics

### State Variables

$v : \text{String}$

$w : \text{String}$

$weight : \mathbb{Z}$

### Access Routine Semantics

Edge( $from, to, w$ ):

- transition:  $v, w, weight := from, to, w$
- output:  $out := self$

weight():

- output:  $out := weight$

from():

- output:  $out := v$

to():

- output:  $out := w$



# City Graph Module

## Template Module

CityGraph

## Uses

Edge

## Syntax

### Exported Types

CityGraph = ?

### Exported Access Programs

| Routine name  | In     | Out         | Exceptions |
|---------------|--------|-------------|------------|
| new CityGraph |        | CityGraph   |            |
| addEdge       | Edge   |             |            |
| adj           | String | seq of Edge |            |

## Semantics

### State Variables

$adj$  : set of tuple of (String, seq of Edge)

### Access Routine Semantics

CityGraph():

- transition:  $adj := \{\}$
- output:  $out := self$

addEdge(e):

- transition:  $adj := \{ \langle str, edges \rangle : \langle String, \text{seq of Edge} \rangle \mid \langle str, edges \rangle \in adj : (str = e.from() \Rightarrow \langle str, edges || [e] \rangle \mid true \Rightarrow \langle str, edges \rangle) \}$

adj(v):

- output:  $out := \{ \langle str, edges \rangle : \langle String, \text{seq of Edge} \rangle \mid \langle str, edges \rangle \in adj \wedge str = v : edges \}$

## EarthquakeBag Module

### Template Module

EarthquakeBag is seq of EarthquakeT

*# Used only for testing time performance against RedBlackBST module.*

*# This module is not part of the actual application.*

## Generic Queue Module

### Generic Template Module inherits Iterable(T)

Queue(T)

### Uses

None

### Syntax

#### Exported Constants

None

#### Exported Types

Queue = ?

#### Internal Types

Node = ?

*# Internal Node type has a link to next item in the queue.*

#### Exported Access Programs

| Routine name | In | Out          | Exceptions             |
|--------------|----|--------------|------------------------|
| Queue        |    | Queue        |                        |
| isEmpty      |    | $\mathbb{B}$ |                        |
| enqueue      | T  |              |                        |
| toString     |    | String       |                        |
| start        |    |              |                        |
| next         |    | T            | NoSuchElementException |

## Semantics

### State Variables

first: Node

last: Node

$n : \mathbb{N}$

$s$ : seq of T

*# For simplification, the linked-node structure is represented by seq of T.*

*#  $s[1]$  is the first Node.*

*#  $s[n]$  is the last Node.*

### State Invariant

None

### Assumptions

None

### Access Routine Semantics

Queue():

- transition: first, last, n := null, null, 0
- output:  $out := self$
- exception: none

isEmpty():

- output:  $out := (n = 0) \Rightarrow True | True \Rightarrow False$
- exception: None

enqueue(item):

- output:  $out := s || item$
- exception: None

toString():

- output:  $out := out || (\forall i : \mathbb{N} | i \in [1..n] : s[i])$
- exception: None

#### Iterator Methods:

$i : \mathbb{N}$

start():

- transition:  $i := 0$
- exception: none

next():

- transition-output:  $i, out := i + 1, s[i]$
- exception:  $(i > n) \Rightarrow \text{NoSuchElementException}$

#### **Considerations**

When an instance of Queue is iterated in a loop, an iterator consisting of these two methods is returned, and the start() method is call initially, and for the successive iterations next() method is call.

## Generic RedBlackBST Module

### Generic Template Module

RedBlackBST(T with Comparable(T), V)

### Uses

Queue

### Syntax

#### Exported Types

RedBlackBST = ?

#### Internal Types

Node = ?

State Variables of Node:

key: Key, lst: seq of V, left: Node, right: Node, color: B, size: N

*# Internal Node type was modified to store a seq of V.*

#### Exported Access Programs

| Routine name | In   | Out          | Exceptions |
|--------------|------|--------------|------------|
| RedBlackBST  |      | RedBlackBST  |            |
| size         |      | N            |            |
| isEmpty      |      | $\mathbb{B}$ |            |
| get          | T    | seq of V     |            |
| put          | T, V |              |            |
| min          |      | T            |            |
| max          |      | T            |            |
| keys         |      | seq of T     |            |
| keys         | T, T | seq of T     |            |
| values       | T, T | seq of V     |            |

## Semantics

### State Variables

root: Node

RED:  $\mathbb{B}$

BLACK:  $\mathbb{B}$

s: set of  $\langle T, V \rangle$

*# For simplification, the linked-node structure is represented by set of  $\langle T, V \rangle$ .*

### State Invariant

RED = True

BLACK = False

### Assumptions

None

### Access Routine Semantics

RedBlackBST():

- transition: None
- output:  $out := self$
- exception: None

size():

- output:  $out := root.size$
- exception: None

isEmpty():

- output:  $out := (root = null) \Rightarrow True | True \Rightarrow False$
- exception: None

get(key):

- output:  $out := L$  where  $\langle x, L \rangle \in s \wedge (x.key = key)$



- exception: None

put(key, val):

- transition:  $s := \{\langle x, L \rangle : \langle T, V \rangle \mid \langle x, L \rangle \in s : (x.key = key \Rightarrow \langle x, L \mid [val] \rangle \mid \text{True} \Rightarrow \langle x, L \rangle)\}$
- exception: None

min():

- output:  $out := \text{smallest key in } s$
- exception: None

max():

- output:  $out := \text{largest key in } s$
- exception: None

keys():

- output:  $out := out \mid (\forall \langle x, L \rangle : \langle T, V \rangle \mid \langle x, L \rangle \in s : x.key)$
- exception: None

keys(lo, hi):

- output:  $out := out \mid (\forall \langle x, L \rangle : \langle T, V \rangle \mid \langle x, L \rangle \in s \wedge lo \leq x.key \leq hi : x.key)$
- exception: None

values(lo, hi):

- output:  $out := out \mid (\forall \langle x, L \rangle : \langle T, V \rangle \mid \langle x, L \rangle \in s \wedge lo \leq x.key \leq hi : L)$
- exception: None

# Search Earthquakes Module

## Module

SearchEarthquakes

## Uses

RedBlackBST, PointT

## Syntax

### Exported Types

None

### Exported Access Programs

| Routine name             | In                                | Out                | Exceptions |
|--------------------------|-----------------------------------|--------------------|------------|
| searchEarthquakeInCircle | RedBlackBST, PointT, $\mathbb{R}$ | seq of EarthquakeT |            |

## Semantics

### State Variables

N/A

### Access Routine Semantics

searchEarthquakeInCircle(bst, location, radius):

- output:  $out := \{e : EarthquakeT \mid e \in bst \wedge location.distanceTo(e.getPointT()) \leq radius : e\}$

# Sort Module

## Module

Sort

## Uses

PointT

## Syntax

### Exported Types

N/A

### Exported Access Programs

| Routine name    | In                         | Out | Exceptions |
|-----------------|----------------------------|-----|------------|
| sortByDistance  | PointT, seq of EarthquakeT |     |            |
| sortByMagnitude | seq of EarthquakeT         |     |            |

## Semantics

### State Variables

N/A

### Access Routine Semantics

sortByDistance(location, eqList):

- transition:  $eqList := eqList$  such that  $\forall(i : \mathbb{N} \mid i \in [0..|eqList| - 2] :$   
 $location.distanceTo(eqList[i].getPointT()) < location.distanceTo(eqList[i+1].getPointT()))$

sortByMagnitude(eqList):

- transition:  $eqList := eqList$  such that  $\forall(i : \mathbb{N} \mid i \in [0..|eqList| - 2] : eqList[i] > eqList[i + 1])$

# Risk Assessment Module

## Template Module

RiskAssessment

## Uses

SearchEarthquakes, GeoCollection, CityGraph, CityPostT

## Syntax

### Exported Types

RiskAssessment = ?

### Exported Access Programs

| Routine name         | In                  | Out          | Exceptions |
|----------------------|---------------------|--------------|------------|
| new RiskAssessment   | RedBlackBST, PointT |              |            |
| getRisk              |                     | $\mathbb{Z}$ |            |
| getCity              |                     | String       |            |
| getFrequency         |                     | $\mathbb{Z}$ |            |
| getMag               |                     | $\mathbb{R}$ |            |
| getPoplationDensity  |                     | $\mathbb{R}$ |            |
| nearestLowerRiskCity | CityGraph           | String       |            |

## Semantics

### State Variables

*earthquakeTree* : RedBlackBST <  $\mathbb{R}$ , EarthquakeT >

*cityProv* : <>

*frequency* :  $\mathbb{Z}$

*averageMag* :  $\mathbb{R}$

*populationDensity* :  $\mathbb{R}$

*rating* :  $\mathbb{Z}$

### Access Routine Semantics

RiskAssessment(bst, location):

- transition: earthquakeTree := bst,  
cityPov := getCityProv(location, SearchEarthquakes.searchEarthquakeInCircle(bst,  
location, 100)),  
frequency := getFrequency(),  
averageMag := getAverageManitude(),  
populationDensity := getPopulation() ,  
rating := OverallRating(frequency, averageMag, populationDensity)

getRisk():

- output: *out* := *rating*

getCity():

- output: *out* := *cityProv*[0]

getFrequency():

- output: *out* := *frequency*

getMag():

- output: *out* := *averageMag*

getPoplationDensity():

- output: *out* := *populationDensity*

nearestLowerRiskCity(graph):

- output: *out* := *min.to()* such that  $min \in graph.adj(getCity) \wedge RiskAssessment(earthquakeTree, getLocation(min.to()).getRisk() < rating \wedge \forall (e : Edge \mid e \in graph.adj(getCity) \wedge RiskAssessment(earthquakeTree, getLocation(e.to()).getRisk() < rating : e.weight \geq min.weight)$

## Local Functions

$\text{getLocation} : \text{String} \rightarrow \text{PointT}$

$\text{getLocation}(\text{city}) \equiv \text{cityPost.getPoint}()$  such that  $\forall(c : \text{CityPostT} \mid c \in \text{seq of CityPostT} \wedge c.\text{getCityName} = \text{city} : \text{cityPost} = c)$

$\text{getCityProv} : \text{PointT}, \text{seq of EarthquakeT} \rightarrow \text{tuple of (String, String)}$

$\text{getCityProv}(\text{location}, \text{eqList}) \equiv \langle \text{eq.getPlace()}, \text{eq.getNameOfProv}() \rangle$  such that  $\forall(e : \text{EarthquakeT} \mid c \in \text{eqList} : \text{location.distanceTo}(e.\text{getPointT}()) \geq \text{location.distanceTo}(\text{eq.getPointT}()))$

$\text{getPopulation} :$

$\text{getPopulation} \equiv \text{city.getPopDensity}()$  such that  $\forall(c : \text{CityT} \mid c \in \text{seq of CityT} \wedge c.\text{getCityName} = \text{cityProv}[0] \wedge c.\text{getProvince} = \text{cityProv}[1] : \text{city} = c)$

$\text{getFrequency} : \text{seq of EarthquakeT} \rightarrow \mathbb{Z}$

$\text{getFrequency}(s) \equiv |s|$

$\text{getAverageMagnitude} : \text{seq of EarthquakeT} \rightarrow \mathbb{R}$

$\text{getAverageMagnitude}(s) \equiv +(e : \text{EarthquakeT} \mid e \in s : e.\text{getMag}()) / |s|$

$\text{frequencyRating} : \mathbb{R} \rightarrow \mathbb{Z}$

$\text{frequencyRating}(\text{frequency}) \equiv (\text{frequency} < 1 \Rightarrow 0 \mid \text{frequency} \geq 1 \wedge \text{frequency} < 10 \Rightarrow 1 \mid \text{frequency} \geq 10 \wedge \text{frequency} < 100 \Rightarrow 2 \mid \text{frequency} \geq 100 \wedge \text{frequency} < 1000 \Rightarrow 3 \mid \text{frequency} \geq 1000 \Rightarrow 4)$

$\text{magnitudeRating} : \mathbb{R} \rightarrow \mathbb{Z}$

$\text{magnitudeRating}(\text{averageMag}) \equiv (\text{averageMag} < 1 \Rightarrow 0 \mid \text{averageMag} \geq 1 \wedge \text{averageMag} < 4 \Rightarrow 1 \mid \text{averageMag} \geq 4 \wedge \text{averageMag} < 6 \Rightarrow 2 \mid \text{averageMag} \geq 6 \wedge \text{averageMag} < 7 \Rightarrow 3 \mid \text{averageMag} \geq 7 \Rightarrow 4)$

$\text{populationdensityRating} : \mathbb{R} \rightarrow \mathbb{Z}$

$\text{populationdensityRating}(\text{populationdensity}) \equiv (\text{populationdensity} < 1000 \Rightarrow 0 \mid \text{populationdensity} \geq 1000 \wedge \text{populationdensity} < 5000 \Rightarrow 1 \mid \text{populationdensity} \geq 5000 \Rightarrow 2)$

$\text{overallRating} : \mathbb{Z}, \mathbb{R}, \mathbb{R} \rightarrow \mathbb{Z}$

$\text{overallRating}(f, a, p) \equiv \text{frequencyRating}(f) + \text{magnitudeRating}(a) + \text{populationdensityRating}(p)$

## View Interface Module

### Interface Module

ViewList

### Uses

PointT

### Syntax

### Exported Types

None

### Exported Access Programs

| Routine name | In                         | Out | Exceptions |
|--------------|----------------------------|-----|------------|
| display      | seq of EarthquakeT, PointT |     |            |

## display by magnitude Module

### Module inherits ViewList

DisplayByMagnitude

### Uses

PointT, Sort

### Syntax

#### Exported Types

None

#### Exported Access Programs

| Routine name | In                         | Out | Exceptions |
|--------------|----------------------------|-----|------------|
| display      | seq of EarthquakeT, PointT |     |            |

### Semantics

#### State Variables

N/A

#### Access Routine Semantics

display(eqList, location):

- print(e.getMag(), e.getColor(), e.getDate().getYear(), e.getPlace())  
for all  $e \in \text{Sort.sortByMagnitude}(\text{eqList})$



## display by distance Module

### Module inherits ViewList

DisplayByDistance

### Uses

PointT, Sort

### Syntax

#### Exported Types

None

#### Exported Access Programs

| Routine name | In                         | Out | Exceptions |
|--------------|----------------------------|-----|------------|
| display      | seq of EarthquakeT, PointT |     |            |

### Semantics

#### State Variables

N/A

#### Access Routine Semantics

display(eqList, location):

- print(location.distanceTo(e.getPointT()), e.getMag(),e.getColor(), e.getDate().getYear(), e.getPlace()) for all  $e \in \text{Sort.sortByMagnitude}(\text{location}, \text{eqList})$

# view risk assessment Module

## Module

ViewRisk

## Uses

RiskAssessment, RedBlackBST, PointT

## Syntax

### Exported Types

None

### Exported Access Programs

| Routine name | In   | Out | Exceptions |
|--------------|--|-----|------------|
| showRisk     | RedBlackBST, seq of CityPostT, PointT, CityGraph |     |            |

## Semantics

### State Variables

N/A

### Access Routine Semantics

display(bst, loc, s, graph):

- print( $ra.getRisk()$ ,  $ra.getCity()$ ,  $ra.getFrequency()$ ,  $ra.getMag()$ ,  $ra.getPoplationDensity()$ ,  $ra.nearestLowerRiskCity(graph)$ ) such that  $ra = RiskAssessment(bst, loc)$

## Local Functions

$initGraph : RiskAssessment, seqof CityPostT, CityGraph \rightarrow CityGraph$

$initGraph(ra, s, graph) \equiv graph.addEdge(e)$  such that  $e = Edge(ra.getCity(), cityPost.getCityName, ra.getCity().getPoint().distanceTo(cityPost.getCityName.getPoint()))$  for all  $cityPost \in s$  and  $ra.getCity().getPoint().distanceTo(cityPost.getCityName.getPoint()) < 100$

# Controller Module

## Module

Controller

## Uses

CSVreader, SearchEarthquakes, ViewList, ViewRisk

## Syntax

### Exported Types

Controller = ?

### Exported Access Programs

| Routine name     | In   | Out | Exceptions |
|------------------|--|-----|------------|
| init             | RedBlackBST, GeoCollection, seq of CityPostT     |     |            |
| search           | RedBlackBST, PointT, $\mathbb{R}$                |     |            |
| updateViewOfList | ViewList   |     |            |
| updateViewOfRisk | RedBlackBST, PointT, seq of CityPostT, CityGraph |     |            |

## Semantics

### State Variables

*location* : PointT

*eqList* : seq of EarthquakeT

### Access Routine Semantics

init(bst, geoCollection, cityPostList):

- transition:  
The states of bst, geoCollection, cityPostLists are modified by accessing the routes of readEarthquakesBST, readPopulation, and readCityPosition in CSVreader module.

search(bst, loc, radius):

- transition: *location* := loc,  
The states of variable eqList is modified by accessing the route of searchEarthquakeInCircle in SearchEarthquakes module.

updateViewOfList(view):

- print the list of earthquakes by accessing the route of display in ViewList module.

updateViewOfRisk(bst, loc, s, graph):

- print the the risk assessment result by accessing the route of showRisk in ViewRisk module.

## 3.2 state machine diagrams

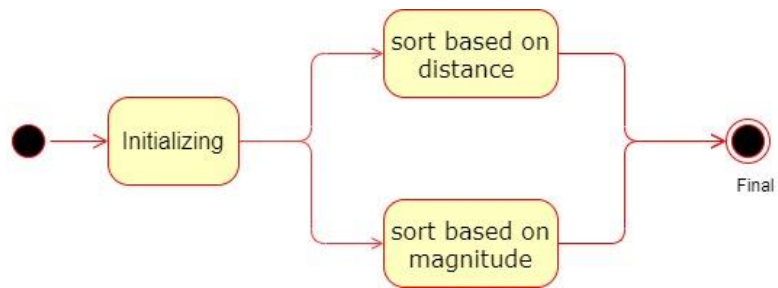


Figure 1 state machine diagram for Sort class

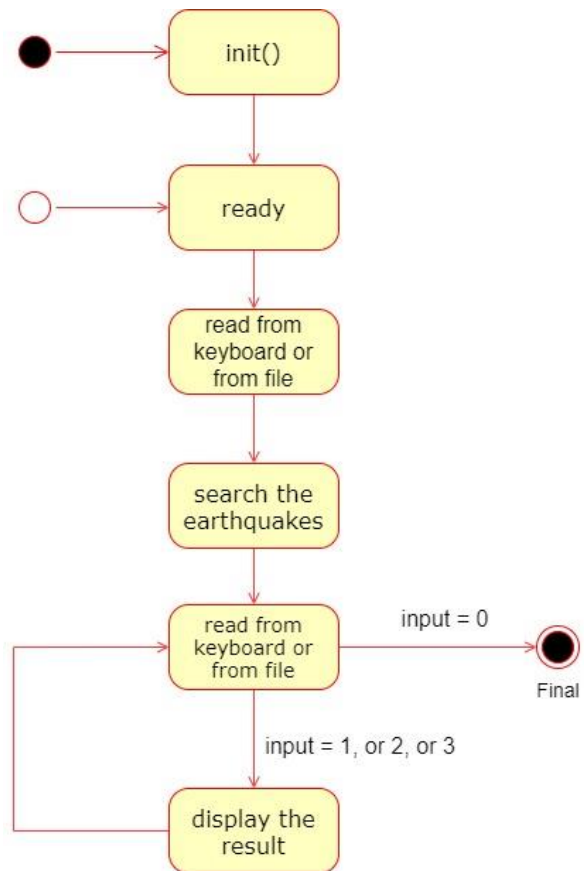


Figure 2 state machine diagram for MCVDemo class

## 4 Internal Review and Evaluation

| class             | requirements   | Details(uses)  |
|-------------------|--|--|
| CSVreader         | Read earthquakes   | Read data to generate color rating then construct <u>EarthquakeT</u> , store them in <u>EarthquakeBag</u>                              |
|                   | read earthquakeBST   | Read data to generate color rating then construct <u>EarthquakeT</u> , store them in <u>RedBlackBST</u>                                |
|                   | read population  | Read data to construct <u>CityT</u> , store them in <u>GeoCollection</u>   |
|                   | read city position   | Read data to construct <u>CityPostT</u> , store them in a sequence   |
| CityPostT         | Accurately represent geographical location of a city       | Use <u>PointT</u> to represent city position   |
| CityT             | Accurately represent city data                             |  |
| EarthquakeT       | Accurately represent earthquake data                       | Use <u>PointT</u> to represent earthquake position, use <u>LocalDateTime</u> to represent earthquake time                              |
| GeoCollection     | Data structure to store cities                             | Store <u>CityT</u>   |
| RedBlackBST       | Data structure to store something                          | Store T using <u>Queue</u>   |
| PointT            | Accurately represent geographical location                 |  |
| Edge              | Represent distance between cities as weight                | Represent connections between two cities   |
| CityGraph         | Represent cities using a directed weighted graph           | Construct a graph made of <u>Edge</u>  |
| EarthquakeBag     | Data structure to store earthquakes                        | Store <u>EarthquakeT</u>   |
| Queue             | An implementation of queue                                 | LinkedList representation of queue   |
| SearchEarthquakes | Search earthquakes within a given radius of input location | Given a position( <u>PointT</u> ) and radius, search required <u>EarthquakeTs</u> in earthquake database( <u>RedBlackBST</u> )         |
| Sort              | Sort earthquakes based on magnitude                        | use <u>quick sort</u> to sort <u>EarthquakeT</u> by magnitude  |
|                   | Sort earthquakes based on distance                         | Use <u>insertion sort</u> to sort <u>EarthquakeT</u> based on distance between earthquake location and given location( <u>PointT</u> ) |

|                    |   |  |
|--------------------|---|--|
| RiskAssessment     | Calculate a risk rating using both earthquake and population data   | Given a position( <u>PointT</u> ) and earthquake database( <u>RedBlackBST</u> ), assess based on earthquake <u>frequency</u> , assess based on earthquake <u>magnitude</u> , assess based on <u>population density(60% accuracy)</u> of the city of nearest earthquake found in <u>GeoCollection</u> , then add up the results |
|                    | find the nearest city that has a lower risk                         | Given a <u>CityGraph</u> , find the nearest city with a lower earthquake risk  |
| ViewList           | Display earthquakes   | Display a sequence of <u>EarthquakeTs</u> based on given location( <u>PointT</u> ) within <u>2</u> seconds( <u>100% accuracy</u> )   |
| DisplayByMagnitude | Display earthquakes by magnitude in descending order                | <u>Sort</u> by magnitude, then displays <u>EarthquakeT</u> information in descending order of magnitude given a sequence of earthquakes and a <u>PointT</u>  |
| DisplayByDistance  | Display earthquakes by distance in ascending order                  | <u>Sort</u> by distance between a position and earthquakes, then displays <u>EarthquakeT</u> information in ascending order of distance given a sequence of earthquakes and a <u>PointT</u>  |
| ViewRisk           | Display risk and earthquake information and nearest lower risk city | Given <u>RedBlackBST</u> and <u>PointT</u> , Display earthquake risk of the point within 4 seconds, display historical earthquake information and population information, display nearest lower risk city got from <u>RiskAssessment</u>   |
| Controller         | Read data   | Use <u>CSVreader</u>   |
|                    | search earthquake   | Use <u>SearchEarthquakes</u>   |
|                    | display earthquake  | Use <u>ViewList</u>  |
|                    | display risk  | Use <u>ViewRisk</u>  |
| Demo               | A sample run  |  |
| ExperimentsSearch  | Compare which algorithm is better                                   |  |

Our project design follows the MVC design pattern. The model is completely separated from view and controller. The model is separated into 5 parts: ADT, search, sort, graph, and riskAssessment. Search and sort use different ADTs, and riskAssessment uses graph and ADTs. The ADT part contains ADTs for position (PointT), city (CityT), city position (CityPostT), and earthquake (EarthquakeT). The search module searches earthquakes within a radius using RedBlackBST. The sort module sorts earthquakes using insertion sort and quick sort. The graph module implements a directed weighted graph between cities. The risk assessment module assesses the risk for a position and finds the nearest city with a lower risk. The view displays earthquake information and risk assessment result using sort and riskAssessment. The controller reads data from csv files, searches earthquakes and updates view. In these modules, only methods that are called by other modules are public, otherwise they are private. Generally speaking, each module has distinct functionalities, while in each module, components are closely connected.

Overall, our program is doing great in encapsulation and modularity. We achieved good reusability and verifiability of code. The quality is ensured by following MIS strictly. The use of BST and quick sort improve the performance, and the use of MVC and strategy design pattern improve the quality and maintainability. The program is not required to be robust, but it is required to be correct and reliable. We do not achieve the goal of making a GUI, but we will work on it in the future to make our program reliable.