# 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

None

### State Variables

None

### State Invariant

None

### Assumptions

None

### 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.
- 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 = | *out* := |
|---|---|---|
| \|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\| $\neq$ 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 | real, real | PointT | RuntimeException |
| getLat | | real | |
| getLong | | real | |
| distance | PointT | real | |
| latFilter | real | $< real, real >$ | |
| equals | PointT | boolean | |

## 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: $out :=< minLat, maxLat >$ 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, real | CityT | |
| getCityName | | string | |
| getProvince | | string | |
| getPopDensity | | real | |
| equals | CityT | boolean | |

## Semantics

### State Variables

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

### Access Routine Semantics

CityT($city, pro, pop$):

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

getCityName():

- output: $out := cityName$

getProvince():

- output: $out := province$

getPopDensity():

- output: $out := popDensity$

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, real, real | 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 | sequence of CityT | |
| getAllCities | | set of tuple of(string, sequence of CityT) | |
| isEmpty | | boolean | |

## Semantics

### State Variables

$s$ : set of tuple of (string, sequence of CityT)

### Access Routine Semantics

add(city):

- transition: $s := \{< str, cities >:< String,$ sequence of CityT $> \mid < str, cities > \in s : (str = getFirstCityLetter(city) \Rightarrow < str, cities || [city] > \mid true \Rightarrow < str, cities > )\}$

isEmpty():

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

getCities(firstLetter):

- output: $out := \{< str, cities >:< String,$ sequence of CityT $> \mid < str, cities > \in s \wedge str = firstLetter : cites\}$

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, $\mathbb{R}, \mathbb{R}, \mathbb{R}, \mathbb{R}$, 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* := date
- exception: None

getColor():

- output: *out* := 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* := (sameDate ∧ samePoint ∧ samePlace ∧ sameDepth ∧ sameMagValue ∧ sameMagType ∧ sameEqClass) ⇒ $True | True \Rightarrow False$
- 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($d$) $\equiv |d.\text{dph} - this.\text{dph}| < 0.0000001$
*#Returns true if the given EarthquakeT object has the same depth value as the current within the tolerance.*

sameMagValue: EarthquakeT $\rightarrow \mathbb{B}$
sameMagValue($d$) $\equiv |d.\text{mag} - this.\text{mag}| < 0.0000001$
*#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 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

$v$ : String
$w$ : 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 | sequence of Edge | |

## Semantics

### State Variables

$adj$ : set of tuple of (string, sequence of Edge)

### Access Routine Semantics

CityGraph():

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

addEdge(e):

- transition: $adj := \{< str, edges >:< String,$ sequence of Edge $> \mid < str, edges > \in adj : (str = e.from() \Rightarrow < str, edges || [e] > \mid true \Rightarrow < str, edges >)\}$

adj(v):

- output: $out := \{< str, edges >:< String,$ sequence of Edge $> \mid < str, edges > \in adj \wedge str = v : edges\}$

# EarthquakeBag Module

## Template Module

EarthquakeBag is seq of EarthquakeT

# 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: $out := s||\text{item}$
- 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: $(i > n) \Rightarrow$ 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 | | $\mathbb{N}$ | |
| isEmpty | | $\mathbb{B}$ | |
| get | T | seq of V | |
| put | T, V | | |
| min | | T | |
| max | | T | |
| keys | | Queue of T | |
| keys | T, T | Queue of T | |
| values | T, T | Queue 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: $s := \{\langle x, L\rangle : \langle T, V\rangle | \langle x, L\rangle \in s : (x.key = key \Rightarrow \langle x, L || [val]\rangle | \text{True} \Rightarrow \langle x, L\rangle)\}$
- exception: None

min():

- output: $out$ := smallest key in $s$
- exception: None

max():

- output: $out$ := largest key in $s$
- exception: None

keys():

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

keys(lo, hi):

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

values(lo, hi):

- output: $out := out || (\forall \langle x, L\rangle : \langle T, V\rangle | \langle x, L\rangle \in s \wedge lo \leq x.key \leq hi : L)$
- 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, $\mathbb{R}$ | sequence of EarthquakeT | |

## Semantics

### State Variables

N/A

### Access Routine Semantics

searchEarthquakeInCircleh(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, sequence of EarthquakeT | | |
| sortByMagnitude | sequence of EarthquakeT | | |

## Semantics

### State Variables

N/A

### Access Routine Semantics

sortByDistance(location, eqList):

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

sortByMagnitude(eqList):

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

# Risk Assessemnt 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

$earthquakeTree : RedBlackBST < Double, EarthquakeT >$
$cityProv :<>$
$frequency : \mathbb{Z}$
$averageMag : \mathbb{R}$
$populationDensity : \mathbb{R}$
$rating : \mathbb{Z}$

**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

getLocation : string $\rightarrow$ PointT
getLocation($city$) $\equiv$ $cityPost.getPoint()$ such that $\forall(c : CityPostT \mid c \in$ sequence of CityPostT $\wedge$ $c.getCityName = city : cityPost = c)$

getCityProv : PointT, sequence of EarthquakeT $\rightarrow$ tuple of (string, string)
getCityProv($location, eqList$) $\equiv <$ $eq.getPlace(), eq.getNameOfProv() >$ such that $\forall(e :$ $EarthquakeT \mid c \in eqList : location.distanceTo(e.getPointT()) \geq location.distanceTo(eq.getPointT()))$

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

getFrequency : sequence of EarthquakeT $\rightarrow \mathbb{Z}$
getFrequency($s$) $\equiv |s|$

getAverageMagnitude : sequence of EarthquakeT $\rightarrow \mathbb{R}$
getAverageMagnitude($s$) $\equiv +(e : EarthquakeT \mid e \in s : e.getMag())/|s|$

frequencyRating : $\mathbb{R} \rightarrow \mathbb{Z}$
frequencyRating($frequency$) $\equiv$ $(frequence < 1 \Rightarrow 0 \mid frequence \geq 1 \wedge frequence <$ $10 \Rightarrow 1 \mid frequence \geq 10 \wedge frequence < 100 \Rightarrow 2 \mid frequence \geq 100 \wedge frequence <$ $1000 \Rightarrow 3 \mid frequence \geq 1000 \Rightarrow 4)$

magnitudeRating : $\mathbb{R} \rightarrow \mathbb{Z}$
magnitudeRating($averageMag$) $\equiv$ $(averageMag < 1 \Rightarrow 0 \mid averageMag \geq 1 \wedge averageMag <$ $4 \Rightarrow 1 \mid averageMag \geq 4 \wedge averageMag < 6 \Rightarrow 2 \mid averageMag \geq 6 \wedge averageMag <$ $7 \Rightarrow 3 \mid averageMag \geq 7 \Rightarrow 4)$

populationdensityRating : $\mathbb{R} \rightarrow \mathbb{Z}$
populationdensityRating($populationdensity$) $\equiv$ $(populationdensity < 1000 \Rightarrow 0 \mid populationdensity \geq$ $1000 \wedge populationdensity < 5000 \Rightarrow 1 \mid populationdensity \geq 5000 \Rightarrow 2)$

overallRating : $\mathbb{Z}, \mathbb{R}, \mathbb{R} \rightarrow \mathbb{Z}$
overallRating($f, a, p$) $\equiv$ $frequencyRating(f)$+$magnitudeRating(a)$+$populationdensityRating(p)$

# View Interface Module

## Interface Module

ViewList

## Uses

PointT

## Syntax

### Exported Types

None

### Exported Access Programs

| Routine name | In | Out | Exceptions |
|---|---|---|---|
| display | sequence 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 | sequence 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 ∈ Sort.sortByMagnitude(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 | sequence 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 ∈ Sort.sortByMagnitude(location, 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, sequence 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 : $RiskAssessment, sequenceofCityPostT, CityGraph \rightarrow CityGraph$
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, sequence of CityPostT | | |
| search | RedBlackBST, PointT, real | | |
| updateViewOfList | ViewList | | |
| updateViewOfRisk | RedBlackBST, PointT, sequence of CityPostT, CityGraph | | |

## Semantics

### State Variables

$location : PointT$
$eqList : sequence of EarthquakeT$

### 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.