Smart Estate Specification

COMPSCI 2XB3 L09 Group 9

April 5, 2019

This Module Interface Specification (MIS) document contains modules, types and methods for implementing Smart Estate.

StateInfo Type Module

Module

StateInfo

Uses

N/A

Syntax

Exported Constants

None

Exported Types

StateInfo = ?
fieldT = {hpi, crime_rate, housing_price}

Exported Access Programs

Routine name	In	Out	Exceptions
new StateInfo	String	StateInfo	none
getState		String	none
getHPI		$\mathbb{R}$	none
setHPI	$\mathbb{R}$		none
getCrimeRate		$\mathbb{R}$	none
setCrimeRate	$\mathbb{R}$		none
getHousingPrice		$\mathbb{R}$	none
setHousingPrice	$\mathbb{R}$		none
toString		String	none

Semantics

State Variables

state: String
hpi: $\mathbb{R}$

crime_rate: $\mathbb{R}$
housing_price: $\mathbb{R}$

State Invariant

None

Assumptions & Design Decisions

- The StateInfo constructor is called for each object instance before any other access routine is called for that object. The constructor can only be called once.
- Once state info is gathered for each StateInfo object methods setHPI, setCrimeRate, and setHousingPrice are only called once.

Access Routine Semantics

new StateInfo(s):

- transition: $state := s$
- output: $out := self$
- exception: none

getState():

- output: $out := state$
- exception: none

getHPI():

- output: $out := hpi$
- exception: none

setHPI(v):

- transition: $hpi := v$
- exception: none

getCrimeRate():

- output: $out := crime\_rate$
- exception: none

setCrimeRate(v):

- transition: $crime\_rate := v$
- exception: none

getHousingPrice():

- output: $out := housing\_price$
- exception: none

getHousingPrice(v):

- transition: $housing\_price := v$
- exception: none

toString():

- output: $out :=$ "$state$: HPI: hpi Crime Rate: $crime\_rate$ Housing Price: $housing\_price$"
- exception: none

PopulateStateInfo Module

Module

PopulateStateInfo

Uses

ReadHPI
ReadCrimeRate
ReadHousingPrices
StateInfo

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
populateStateInfo		seq of StateInfo	none

Semantics

State Variables

states: seq of StateInfo
state_names: seq of String = ["Alabama","Alaska", ... , "Wyoming"]

State Invariant

None

Assumptions & Design Decisions

- The result of populateStateInfo must be stored in a StateInfo list of 50 length.

Access Routine Semantics

populateStateInfo():

- transition: initStates(); populateHPI(); populateCrimeRate(); populateHousingPrice();

- output: $out := states$
- exception: None

Local Functions

initStates() $\equiv$ $states$:=($\forall s$:String $|s \in state\_names \ . \ s =$ StateInfo(s))

populateHPI() $\equiv$ $states$:=
($\forall i$:int $|0 \leq i \leq 50$. $states[i]$.setHPI(ReadHPI.read_data("data/hpi.csv").value()))

populateCrimeRate() $\equiv$ $states$:=
($\forall i$:int $|0 \leq i \leq 50$. $states[i]$.setCrimeRate(ReadCrimeRate.CRList().value()))

populateHousingPrice() $\equiv$ $states$:=
($\forall i$:int $|0 \leq i \leq 50$. $states[i]$.setHousingPrice(ReadHousingPrices.
readPrices("data/housingPrices.csv").value()))

Binary Search Module

Module

binSearch

Uses

StateInfo
Sort

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
binSearch	seq of StateInfo, fieldT, $\mathbb{R}$	StateInfo	none
binSearch	seq of StateInfo, String	StateInfo	none

Semantics

Assumptions & Design Decisions

-

Access Routine Semantics

binSearch($arr, field, key$):

- output: $out := arr[i]$ such that
  $isSorted(arr, field) \wedge$
  $(key \in \{arr.field\} \implies arr[i].field = key) \wedge$
  $(key < \min(\{arr.field\}) \implies arr[i] = arr[0]) \wedge$
  $(key > \max(\{arr.field\}) \implies arr[i] = arr[arr.length - 1]) \wedge$
  $(key \notin \{arr.field\} \implies arr[i-1].field < arr[i].field = key < arr[i+1].field)$
- exception: none

binSearch(arr, key):

- output: $out := arr[i]$ such that $arr[i].state = key$
- exception: none

ReadCrimeRate Module

Module

ReadCrimeRate

Uses

Pair

Syntax

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
load_crime_data	s : string		

Semantics

Environment Variables

crime_rate_data: File listing crime rate data

State Variables

None

State Invariant

None

Assumptions

The input file will match the given specification.

Access Routine Semantics

load_crime_data(s)

- transition: read data from the file crime_rate_data associated with the string s. Use this data to create an array of Pairs, which house the name of a state along with the average number of violent crimes per capita over 49 years for every 100,000 person.

  The csv file has the following format, where *year*, *population*, total number of *violent crime*, followed by a breakdown of the number of violent crimes into sub categories including murder, robbery, aggravated assault, etc. which is not used in the computation of the overall project. This is split by a 5 wide horizontal gap separating each state's independent statistics.

$$\begin{array}{cccc} year_0, & population_0, & violent_crimes_0, & ... \\ year_1, & population_1, & violent_crimes_1, & ... \\ year_2, & population_2, & violent_crimes_2, & ... \\ ..., & ..., & ..., & ... \\ year_{m-1}, & population_{m-1}, & violent_crimes_{m-1}, & ... \end{array} \quad (1)$$

- exception: FileNotFoundException

Module

Sort

Uses

binSearch BreadthFirstSearch Integration

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
sort	sequence of StateInfo, fieldT		none
sort	sequence of StateInfo		none

Semantics

State Variables

None

State Invariant

None

Assumptions & Design Decisions

- Sort is called on a sequence of StateInfo and category by which the sequence is sorted is also indicated when sort is called.

Access Routine Semantics

sort(a, intent) :

- transition: $a := sort(a, 0, |a| - 1, intent)$
- exception: none

sort(a):

- transition: $a := sort(a, 0, |a| - 1)$
- exception: none

Local Functions

sort$(a, lo, hi, intent) \equiv$a := $(\forall i LT | i \in a : a[i-1] ¡ a[i])$

sort$(a, lo, hi) \equiv$a := $(\forall i LT | i \in a : a[i-1] ¡ a[i])$

partition$(a, lo, hi, intent) \equiv j \implies a[lo] ¡= a[j] ¡= a[hi]$

partition$(a, lo, hi) \equiv j \implies a[lo] ¡= a[j] ¡= a[hi]$

less$(v, w, intent) \equiv v < w$
less$(v, w) \equiv v < w$

exch$(a, i, j) \equiv a[i], a[j] := a[j], a[i]$

Graph Type Module

Module

Graph

Uses

BreadthFirstSearch
Integration

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
new Graph	$\mathbb{N}$		none
genGraph		*Graph*	none
V	$\mathbb{N}$		none
E	$\mathbb{N}$		none
addEdge	String, String		none
adj	$\mathbb{N}$	sequence of $Bag < String >$	none

Semantics

State Variables

- V: $\mathbb{N}$
- *states*: sequence of Strings

State Invariant

None

Assumptions & Design Decisions

- In order to generate a graph the represents the US, genGraph is called instead of the constructor.

Access Routine Semantics

new Graph(V):

- transition: $V, E := V, 0$
- exception: none

genGraph():

- output: $out :=$ Graph representing the US map
- exception: none

V():

- output: $out := V$
- exception: none

E():

- output: $out := E$
- exception: none

addEdge(v, w):

- transition: $E, adj[v.index()] := E++, adj[v.index()].add(w)$
- exception: none

adj(v):

- output: $out := adj[v]$
- exception: none

Breadth First Search Module

Module

BreadthFirstSearch

Uses

Integration

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
BreadthFirstSearch	$Graph, String$	$sequence of strings$	none
bfs		$Graph, String\ sequence of strings$	none
getStateInfo	$sequence of strings$	$sequence of strings$	none
neighbourStates	String	$sequence of strings$	none

Semantics

State Variables

- $path$: $sequence of strings$

State Invariant

None

Assumptions & Design Decisions

- In order to implement breadth first search on a graph representing the U.S. neighbourStates is called

Access Routine Semantics

BreadthFirstSearch(G, s) :

- output: out := sequence of strings
- exception: none

bfs(G, s) :

- output: out := sequence of strings
- exception: none

getStateInfo($neighbours$) :

- output: out := sequence of strings
- exception: none

neighbourStates($startState$):

- output: out := sequence of strings
- exception: none