Assignment 3, Part 1, Specification

SFWR ENG 2AA4

March 2, 2020

This Module Interface Specification (MIS) document contains modules, types and methods for implementing a generic 2D sequence that is instantiated for both land use planning and for a Discrete Elevation Model (DEM).

In applying the specification, there may be cases that involve undefinedness. We will interpret undefinedness following [**?**]:

If $p : \alpha_1 \times .... \times \alpha_n \rightarrow \mathbb{B}$ and any of $a_1, ..., a_n$ is undefined, then $p(a_1, ..., a_n)$ is False. For instance, if $p(x) = 1/x < 1$, then $p(0) =$ False. In the language of our specification, if evaluating an expression generates an exception, then the value of the expression is undefined.

[The parts that you need to fill in are marked by comments, like this one. In several of the modules local functions are specified. You can use these local functions to complete the missing specifications. —SS]

[As you edit the tex source, please leave the `wss` comments in the file. Put your answer **after** the comment. This will make grading easier. —SS]

Land Use Type Module

Module

LanduseT

Uses

N/A

Syntax

Exported Constants

None

Exported Types

Landtypes = {R, T, A, C}

//R stands for Recreational, T for Transport, A for Agricultural, C for Commercial

Exported Access Programs

Routine name	In	Out	Exceptions
new LanduseT	Landtypes	LanduseT	

Semantics

State Variables

landuse: Landtypes

State Invariant

None

Access Routine Semantics

new LandUseT(t):

- transition: *landuse* := t

- output: *out* := self
- exception: none

Considerations

When implementing in Java, use enums (as shown in Tutorial 06 for ElementT).

Point ADT Module

Template Module inherits Equality(PointT)

PointT

Uses

N/A

Syntax

Exported Types

[What should be written here? —SS]
PointT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
PointT	$\mathbb{Z}$, $\mathbb{Z}$	PointT	
row		$\mathbb{Z}$	
col		$\mathbb{Z}$	
translate	$\mathbb{Z}$, $\mathbb{Z}$	PointT	

Semantics

State Variables

r: [What is the type of the state variables? —SS]
c: [What is the type of the state variables? —SS]
r: $\mathbb{Z}$
c: $\mathbb{Z}$

State Invariant

None

Assumptions

The constructor PointT is called for each object instance before any other access routine is called for that object. The constructor cannot be called on an existing object.

Access Routine Semantics

PointT(row, col):

- transition: [What should the state transition be for the constructor? —SS]
  transition: $r, c = row, col$
- output: $out := self$
- exception: None

row():

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

col():

- [What should go here? —SS]
  output: $out := c$
- exception: None

translate(Δr, Δc):

- [What should go here? —SS]
  output: $out :=$ new PointT($(r + \Delta r)$, $(c + \Delta c)$)
- exception: [What should go here? —SS]
  expection: None

Generic Seq2D Module

Generic Template Module

Seq2D(T)

Uses

PointT

Syntax

Exported Types

Seq2D(T) = ?

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
Seq2D	seq of (seq of T), $\mathbb{R}$	Seq2D	IllegalArgumentException
set	PointT, T		IndexOutOfBoundsException
get	PointT	T	IndexOutOfBoundsException
getNumRow		$\mathbb{N}$	
getNumCol		$\mathbb{N}$	
getScale		$\mathbb{R}$	
count	T	$\mathbb{N}$	
countRow	T, $\mathbb{N}$	$\mathbb{N}$	
area	T	$\mathbb{R}$	

Semantics

State Variables

s: seq of (seq of T)
scale: $\mathbb{R}$
nRow: $\mathbb{N}$
nCol: $\mathbb{N}$

State Invariant

None

Assumptions

- The Seq2D(T) constructor is called for each object instance before any other access routine is called for that object. The constructor can only be called once.
- Assume that the input to the constructor is a sequence of rows, where each row is a sequence of elements of type T. The number of columns (number of elements) in each row is assumed to be equal. That is each row of the grid has the same number of entries. $s[i][j]$ means the ith row and the jth column. The 0th row is at the top of the grid and the 0th column is at the leftmost side of the grid.

Access Routine Semantics

Seq2D(S, scl):

- transition: [Fill in the transition. —SS]
  transition: s, scale, nRow, nCol := S, scl, $|S|, |S[0]|$
- output: $out := self$
- exception: [Fill in the exception. One should be generated if the scale is less than zero, or the input sequence is empty, or the number of columns is zero in the first row, or the number of columns in any row is different from the number of columns in the first row. —SS]
  exc := $(\text{scl} < 0) \vee (|S| = 0) \vee (|S[0]| = 0) \vee \neg(\forall\ \text{i} \mid \text{i} \in [1..\text{nRow} - 1] : |\text{S[i]}| = |\text{S[0]}|) \Rightarrow$ IllegalArgumentException

set(p, v):

- transition: [? —SS]
  transition: $s[p.\text{row}()][p.\text{col}()] := v$
- exception: [Generate an exception if the point lies outside of the map. —SS]
  expection: exc := $\neg(\text{validPoint(p)}) \Rightarrow$ IndexOutOfBoundsException

get(p):

- output: [? —SS]
  output: $out := s[p.\text{row}()][p.\text{col}()]$

- exception: [Generate an exception if the point lies outside of the map. —SS]
  expection: exc := ¬(validPoint(p)) ⇒ IndexOutOfBoundsException

getNumRow():

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

getNumCol():

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

getScale():

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

count(t: T):

- output: [Count the number of times the value t occurs in the 2D sequence. —SS]
  output: *out* := $(+ r : \mathbb{N} \mid r \in [0..|s| - 1] : \text{countRow}(t, r))$
- exception: None

countRow(t: T, $i : \mathbb{N}$):

- output: [Count the number of times the value t occurs in row i. —SS]
  output: *out* := $(+ j : \mathbb{N} \mid (j \in [0..|s[i]| - 1] \wedge s[i][j] = t) : 1)$
- exception: [Generate an exception if the index is not a valid row. —SS]
  exception: exc := ¬(validRow(i)) ⇒ IndexOutOfBoundsException

area(t: T):

- output: [Return the total area in the grid taken up by cell value t. The length of each side of each cell in the grid is scale. —SS]
  output: *out* := count(t) ∗ $scale^2$
- exception: None

Local Functions

validRow: $\mathbb{N} \rightarrow \mathbb{B}$
[returns true if the given natural number is a valid row number. —SS]
validRow(r) $\equiv r <$ getNumRow()

validCol: $\mathbb{N} \rightarrow \mathbb{B}$
[returns true if the given natural number is a valid column number. —SS]
validCol(c) $\equiv c <$ getNumCol()

validPoint: PointT $\rightarrow \mathbb{B}$
[Returns true if the given point lies within the boundaries of the map. —SS]
validPoint(p) $\equiv$ validCol(p.col()) $\wedge$ validRow(p.row())

LanduseMap Module

Template Module

[Instantiate the generic ADT Seq2D(T) with the type LanduseT —SS]
LanduseMap is Seq2D(LanduseT)

DEM Module

Template Module

DemT is Seq2D($\mathbb{Z}$)

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
total		$\mathbb{Z}$	
max		$\mathbb{Z}$	
ascendingRows		$\mathbb{B}$	

Semantics

Access Routine Semantics

total():

- output: [Total of all the values in all of the cells. —SS]
  output: *out* := $(+\ r, c : \mathbb{N} \mid (\text{validRow}(r) \wedge \text{validCol}(c)) : \text{s[r][c]})$
- exception: None

max():

- output: [Find the maximum value in the 2d grid of integers —SS]
  output: *out* := x such that $(\forall\ \text{r, c} : \mathbb{N} \mid \text{validRow}(r) \wedge \text{validCol}(c) : \text{x} \geq \text{s[r][c]}) \wedge \text{count}(x) > 0$
- exception: None

ascendingRows():

- output: [Returns True if the sum of all values in each row increases as the row number increases, otherwise, returns False. —SS]
  output: *out* := $(\forall\ r : \mathbb{N} \mid r \in [0..|s| - 2] : (\text{rowSum}(r) < \text{rowSum}(r{+}1)))$
- exception: None

Local Functions

validRow: $\mathbb{N} \rightarrow \mathbb{B}$
[returns true if the given natural number is a valid row number. —SS]
validRow(r) $\equiv r <$ getNumRow()

validCol: $\mathbb{N} \rightarrow \mathbb{B}$
[returns true if the given natural number is a valid column number. —SS]
validCol(c) $\equiv c <$ getNumCol()

rowSum: $\mathbb{N} \rightarrow \mathbb{N}$
Returns the sum of the given row natural number
rowSum(r) $\equiv (+\ c : \mathbb{N} \mid (c \in [0..\text{nCol} - 1] : s[r][c])$

Critique of Design

[Write a critique of the interface for the modules in this project. Is there anything missing? Is there anything you would consider changing? Why? One thing you could discuss is that the Java implementation, following the notes given in the assignment description, will expose the use of ArrayList for Seq2D. How might you change this? There are repeated local functions in two modules. What could you do about this? —SS]
I don't think anything is missing as we do not know what is required and what is not required. One thing I would considering changing the repeated local function validRow and validCol. I would get rid of those functions from DemT but keep it in Seq2D because DemT inherits from Seq2D. So no need to overwrite the methods as they perform the same task as before. Instead of using an ArrayList for the Seq2D use an Array instead.

In addition to your critique, please address the following questions:

1. The original version of the assignment had an Equality interface defined as for A2, but this idea was dropped. In the original version Seq2D inherited the Equality interface. Although this works in Java with the LanduseMapT, it is problematic for DemT. Why is it problematic? (Hint: DEMT is instantiated with the Java type Integer.)

2. Although Java has several interfaces as part of the standard language, such as the Comparable interface, there is no Equality interface. Instead equals is provided through inheritance from Object. Why do you think the Java language designers decided to use inheritance for equality, instead of providing an interface?

3. The qualities of good module interface push the design of the interface in different directions. Why is it rarely possible to achieve a module interface that simultaneously is essential, minimal and general?

Answers

1. DemT inheriting the equality interface would be problematic because you would be using the == operator to compare the two values. Integer in Java is an object type, so using == would mean comparing the two Integer objects' memory location. Though if the Integer was between -128 and 127 then == would work but you can assume the number will be between this range. The == works with LanduseMapT because there you will be comparing the enum which are constant values and they will have the same memory location for the same type enums. Using == on LanduseMapT would be safe but not for DemT. Source

2. The reason I think Java language designers decided to use inheritance for equality, instead of providing an interface because all objects can be checked for equality. You can check if two objects are the same in terms of some criteria or have the same memory location (default equals method for objects). An interface like Comparable means that different objects can be compared with each other, which makes sense to implement an interface for it. Since not all objects can be compared but those can just use the Comparable interface. However, it makes sense to have Equality as inheritance from Object because equality is can be applied to members of the same object.

3. It is rarely possible to achieve a module interface that simultaneously is essential, minimal and general because essential means to omit unnecessary features, minimal means to avoid access routines that have two independent services and general means to have multiple purposes for the module. If you want a module to be general then it might be hard to make it also essential because essential focuses on what is important for that specific situation. Having essential and minimal both in an interface might be tough as you might want multiple things done in the access program as per the client/MIS but this will break the minimal quality.