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

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] $r, c := row, col$
- output: $out := self$
- exception: None

row():

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

col():

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

translate(Δr, Δc):

- [What should go here? —SS] $out :=$ PointT$(r + \Delta r, c + \Delta c)$
- exception: [What should go here? —SS] 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] $s, \text{scale}, \text{nCol}, \text{nRow} := S, \text{scl}, |S[0]|, |S|$
- 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] $exc :=$ (scale $\leq 0 \vee |S| = 0 \vee |S[0]| = 0 \Rightarrow$ IllegalArgumentException$|$ $\neg\forall(l : \text{seq of T} | l \in S : |l| = |S[0]|) \Rightarrow$ IllegalArgumentException)

set(p, v):

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

get(p):

- output: [? —SS]$out := s[p.r][p.c]$
- exception: [Generate an exception if the point lies outside of the map. —SS]$exc :=$ ($\neg$validPoint(p) $\Rightarrow$ 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]$out := +(i, j : \mathbb{N} | \text{validRow}(i) \wedge \text{validCol}(j) \wedge s[i][j] = t : 1)$
- exception: None

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

- output: [Count the number of times the value t occurs in row i. —SS] $out := +(p : \text{PointT} | p \in \text{s}[i] \wedge s[p.r][p.c] = t : 1)$
- exception: [Generate an exception if the index is not a valid row. —SS] $exc := (\neg\text{validRow}(i) \Rightarrow \text{IndexOutOfBoundsException})$

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] $out := +(p : \text{PointT} | p \in \text{s}[i] \wedge s[p.r][p.c] = t : (p.r \cdot scale) \cdot (p.c \cdot scale))$
- exception: None

Local Functions

validRow: $\mathbb{N} \to \mathbb{B}$
[returns true if the given natural number is a valid row number. —SS]validRow$(i) \equiv 0 \leq i \leq (\text{nRow} - 1)$

validCol: $\mathbb{N} \to \mathbb{B}$
[returns true if the given natural number is a valid column number. —SS]validCol$(j) \equiv 0 \leq j \leq (\text{nCol} - 1)$

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

LanduseMap Module

Template Module

[Instantiate the generic ADT Seq2D(T) with the type LanduseT —SS]
LanduseMapT 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]
  $out := +(i : \mathbb{N}, j : \mathbb{N} \mid i \in [0, ...nRow - 1] \ \wedge \ j \in [0, ...nCol - 1] : s[i][j].r)$
- exception: None

max():

- output: [Find the maximum value in the 2d grid of integers —SS]
  $out := +(i : \mathbb{N}, j : \mathbb{N} \mid i \in [0, ...nRow - 1] \wedge j \in [0, ...nCol - 1] : (max : \mathbb{N} \mid max < s[i][j] : max := s[i][j]))$
- 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]
  $out := (i : \mathbb{N}, j : \mathbb{N} \mid i \in [1, ...nRow - 1] \wedge j \in [1, ...nCol - 1] : ((s[i-1][j] < s[i][j] \rightarrow true) \mid (s[i-1][j] > s[i][j] \rightarrow false))$
- exception: None

Local Functions

validRow: $\mathbb{N} \to \mathbb{B}$
[returns true if the given natural number is a valid row number. —SS]validRow$(i) \equiv 0 \leq i \leq (\text{nRow} - 1)$

validCol: $\mathbb{N} \to \mathbb{B}$
[returns true if the given natural number is a valid column number. —SS]validCol$(j) \equiv 0 \leq j \leq (\text{nCol} - 1)$

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]

The interface overall was designed quite well but there are few things that stood out to me that could be changed. One being that of the get and set methods of the Seq2D. They both seem to doing more than one thing and overlap as doing similar actions throughout the program. For example, get method retrieves the data while the set method also does this with the addition of modifying the data to update it. They're both similar in this sense and for this reason, minimality is violated for this design specification which stood out to me. In addition to this, the Java implementation currently exposes ArrayList for Seq2D class. Generally, Java-v8 interfaces can only have public abstract methods and variables without the capability of using protected or private keywords. This was changed in Java-v9 but we are currently using Java-v8. If this was an interface, we could get around this by creating a class inside the Seq2D class which contains private methods and variables which effectively encapsulates the data and prevents it from being exposed. Essentially we are making the ArrayList private in a way that doesn't impact the rest of the Seq2D class. For the third question asking what to do with repeated local functions, I would just create a separate abstract class that contains these common methods. This new class could then be inherited into the Seq2D class by extending it which allows for easy access to the common previously local methods. Since interfaces and abstract classes are public static, this means we don't need to instantiate them in any way which is why we can easily extend the classes over and access the helper methods inside.

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.)

   The reason it is NOT problematic for LanduseMapT is because the LanduseMapT is a generic implementation of SeqT interface which means it can implement the interface without declaring any methods of certain types. So essentially a class that implements the SeqT interface but is empty inside. On the other hand, DEM is instantiated with Integer type meaning it is not generic and uses methods of a

certain type from the beginning, instead of adapting dynamically.

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?

   Equality comparisons are likely to be used throughout various objects in Java programs. By having it built into Java generic class objects, it significantly makes it more efficient and less time consuming as the methods can be inherited from the object rather than implementing the interface each time it is needed. This reduces redundant code and makes the program more essential. In addition, the other components of the class can also be implemented too which is a added bonus to being able to implement the equality method. This is why having equality inherited by an object is more beneficial compared to inheriting from interfaces as it is a task that is likely to occur often.

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?

The qualities of a good module interface depends upon the decisions that are needed to benefit the API to do certain tasks. It is sometimes necessary to violate the qualities of good module interface design if it means the API will behave more "nicely" and reduce the hassle when using it. Essentiality is great in removing redundant code and having minimality is also important so that a module doesn't do more than one task that another module might also do. However, sometimes these properties may need to be violated. For example, if a stack needs to have its top element popped off and returned, then it no longer makes sense to have a peak() function to read what the top value is as the pop() function both modifies the stack and returns the popped element. In this case, the interface is no longer minimal as the pop() function modifies the stack and returns the top element which similar to what the peak() function does by reading what the top element is and returning it to the client. The API could be designed to just pop off the top element in the stack and not return the value. If it were designed this way, then yes, it is minimal because each method is doing one specific task without overlapping and doing multiple tasks. It really is up to the developer and their intuition on whether their interface design needs certain qualities as sometimes not having all the qualities can be beneficial in that the API can be easier and less annoying to use.