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: $out := \text{self}$
- 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

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: $\mathbb{Z}$
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: $r := row, c := col$
- output: $out := self$
- exception: None

row():

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

col():

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

translate(Δr, Δc):

- output: $out :=$ PointT($r + \Delta r$, $c + \Delta c$)
- exception: 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 (note that the list does not enforce an *order* in which the transitions occur, only the transitions that must occur):
  1. $s := S$
  2. $scale := scl$
  3. $nRow := |S|$
  4. $nCol := |S[0]|$
- output: $out := self$
- exception:
  exc := (scl < 0) $\vee$ (S = <>) $\vee$ ($(|S[0]| = 0)$) $\vee$ $(\exists(x : \text{Seq of T} \; |x \in S[1..|S|-1] : |x| \neq |S[0]|))$ $\implies$ IllegalArgumentException

set(p, v):

- transition: $s[p.row()][p.col()] = v$
- exception:
  exc := (p.row() >= nRow) $\vee$ (p.col() >= nCol) $\vee$ (p.row() < 0) $\vee$ (p.col() < 0) $\implies$ IndexOutOfBoundsException

get(p):

- output: $out := s[p.row()][p.col()]$
- exception:
  exc := (p.row() >= nRow) $\vee$ (p.col() >= nCol) $\vee$ (p.row() $<$ 0) $\vee$ (p.col() $<$ 0)
  $\implies$ IndexOutOfBoundsException

getNumRow():

- output: out := nRow
- exception: None

getNumCol():

- output: out := nCol
- exception: None

getScale():

- output: out := scale
- exception: None

count(t: T):

- output: out := $(+i : \mathbb{N} | i \in [0..|s| - 1] : \text{countRow}(t, i))$
- exception: None

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

- output: out := $(+x : \text{T} \; | x \in s[i] \wedge x = t : 1)$
- exception: exc := $\neg$ validRow(i) $\implies$ IndexOutOfBoundsException

area(t: T):

- output: out := count(t) $*(scale * scale)$
- exception: None

Local Functions

validRow: $\mathbb{N} \rightarrow \mathbb{B}$
validRow$(r) \equiv r \geq 0 \wedge (r <$ nRow$)$

validCol: $\mathbb{N} \rightarrow \mathbb{B}$
validCol$(c) \equiv (c \geq 0) \wedge (c <$ nCol$)$

validPoint: PointT $\rightarrow \mathbb{B}$
validPoint$(p) \equiv$ validCol(p.col()) $\wedge$ validRow(p.row())

LanduseMap Module

Template Module

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 : out := $+(x, y : \mathbb{N} |$ validRow$(x) \wedge$ validCol(y) : $s[x][y])$
- exception: None

max():

- output: out := M such that $\forall(x :$ Seq of $\mathbb{Z}$ $|x \in s : \forall(y : \mathbb{Z}|y \in x : M \geq y))$
- exception: None

ascendingRows():

- output:out := $\forall(i : \mathbb{N}|i \in [0..|s| - 2] : sum(s[i]) < sum(s[i + 1]))$
- exception: None

Local Functions

validRow: $\mathbb{N} \rightarrow \mathbb{B}$
validRow$(n) \equiv (n \geq 0) \wedge (n <$ nRow$)$

validCol: $\mathbb{N} \rightarrow \mathbb{B}$
$\text{validCol}(c) \equiv (c \geq 0) \wedge (c < \text{nCol})$

sum: Seq of $\mathbb{Z} \rightarrow \mathbb{Z}$
$\text{sum}(s) \equiv (+x : \mathbb{Z} | x \in s : x)$

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 would change the specification to avoid stating to use an ArrayList and provide no specific implementation details. This is to free the programmer from any implementation constraints and let that be their own design decision as long as the specification is satisfied. For the repeated local functions (validRow() and validCol()) since they have the exact same implementation I would make them a publicly accessible routine in Seq2D and DemT can then inherit the method. This can also potentially make Seq2D easier to use and require a client to write less code. For example, suppose the client wants to traverse the 2D sequence horizontally and diagonally then with the current implementation they can use a loop and a set of variables that should be within the bounds of the 2D sequence. Although they can get the number of rows and columns with getNumRow() and getNumCol() respectively, they have to write out the full logic for checking if their variables are within bounds which may look something like:

```
Assuming ''maze'' is an instance of Seq2D of some type
while(i >= 0 && i < maze.getNumRow() && j >= 0 && j < maze.getNumCol()){
//traverse/access maze[i][j] etc.
}
```

The above loop condition gets messy and complicated. But with publicly accessible methods validRow() and validCol() it can be cleanly written as:

```
Assuming ''maze'' is an instance of Seq2D of some type
while(validRow(i) && validCol(i) ){
//traverse/access maze[i][j] etc.
}
```

Furthermore, if validRow() is publicly accessible then a client can use to verify a row number before calling countRow(). Same idea for set(), a client can validate their row and column values before passing a point object to set(), adding additional safety to the software usage and thus prone to less errors/exceptions.

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.)

   In Java, the type Integer is a reference type which is different from the primitive type int. Integer inherits from Object and as a result equality of two Integers is defined as the equality of their references, i.e two Integer values are the same if they point to the same memory location. For LanduseMapT it is instantiated with type LandUseT which is an enumeration. enums in Java is not an object but rather a special data type. Two enum objects are considered equal if either they reference the same memory location or their values are equal. Consider this code:

   ```
   enum Color {
        Red, Green, Blue;
   }
   public class MyClass {
        public static void main(String args[]) {
          Integer p = new Integer(5);
          Integer q = new Integer(5);
          System.out.println(q.equals(p));
          System.out.println(q == p);

          Color c1 = Color.Green;
          Color c3 = Color.Green;
          System.out.println(c3.equals(c1));
          System.out.println(c3 == c1);
       }
   }
   ```

   The output of the above code is :
   true
   false
   true
   true

   Notice that for the Integer object, if the two objects have the same value but are instantiated separately (i.e they reference different memory locations) then they are not considered equal. This requires explicitly using .equals()

However, for the enum object even if two objects have the same value but reference different memory locations they are still equal. Thus, it is problematic to inherit a Equality interface for DemT due to definition of Equality for Integer versus enum in Java.

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?

   I believe this choice was mad due to the nature of equality of some types in Java. Generally, two objects are considered equal if they reference the same memory location but certain types do not have to meet this requirement(e.g enums as discussed above). So to avoid ambiguity when it comes to what "Equality" is defined as, .equals() is provided through inheritance from Object since this will always compare memory references and this allows for a standard definition of "Equality" for all the different types/objects in Java.

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?

   A module is essential if it has no redundant features. This means there are no methods/routines, state variables and invariant that can be removed. A module is minimal if for each routine we have as few state transitions as possible and different services/transitions are independent of one another. A module is general if it is designed without a specific use in mind but rather address a broader domain. For example, consider the scipy library in python. It is not designed to allow users to do matrix operations or solve systems of differential equations. It addresses the general problem of $emphScientificComputing$ and working with matrices and differential equations falls under these. What makes the scipy library general is that there is no specific use for an ODE solver. A client may use it solve a spring-mass system, an electric circuit or even calculate the position of planets but scipy does not enforce any of this, it simply requires a generic set of parameters such as initial conditions, a list of functions etc.

   It is rarely possible to achieve a module interface that simultaneously meets the criteria of being essential, minimal and general.