Assignment 4, Two Dots game Specification

SFWR ENG 2AA4

April 2, 2020

This Module Interface Specification (MIS) document contains modules, types and methods for implementing a game of Two Dots

Color Module

Module

Color

Uses

N/A

Syntax

Exported Constants

None

Exported Types

Color = {R, G, B, P, Y}

//R stands for Red, G for green, B for blue, P for Purple, Y for yellow

Exported Access Programs

Routine name	In	Out	Exceptions
randomColor		Color	

Semantics

State Variables

colors: color

State Invariant

None

Access Routine Semantics

randomColor():

- transition: none

- output: *out* := randomVal()
- exception: none

Local Functions

randomVal(): Color
randomVal() $\equiv (i = 0 \implies \text{R} \mid i = 1 \implies \text{G} \mid i = 2 \implies \text{B} \mid i = 3 \implies \text{P} \mid i = 4 \implies \text{Y})$
Where i is a uniformly-distributed random number in the range $0 \leq i \leq 4$

Point ADT Module

Template Module

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

Generic Board Module

Generic Template Module

Board(T)

Uses

PointT

Syntax

Exported Types

Board(T) = ?

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
Board	$\mathbb{N}, \mathbb{N}$	Board	IllegalArgumentException
set	PointT, T		IndexOutOfBoundsException
get	PointT	T	IndexOutOfBoundsException
getNumRow		$\mathbb{N}$	
getNumCol		$\mathbb{N}$	

Semantics

State Variables

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

State Invariant

None

Assumptions

- The Board(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

Board(*row*, *col*):

- transition (note that the list does not enforce an *order* in which the transitions occur, only the transitions that must occur):
  1. $nRow := row$
  2. $nCol := col$
- output: $out := self$
- exception:
  exc := (row ≤ 0) $\vee$ (col ≤ 0) $\implies$ IllegalArgumentException

set(p, v):

- transition: $s[p.row()][p.col()] = v$
- exception:
  $\neg$ validPoint(p) $\implies$ IndexOutOfBoundsException

get(p):

- output: $out := s[p.row()][p.col()]$
- exception:
  $\neg$ validPoint(p) $\implies$ IndexOutOfBoundsException

getNumRow():

- output: $out :=$ nRow

- exception: None

getNumCol():

- output: *out* := nCol
- 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())

TwoDotsBoard Module

Template Module

TwoDotsBoard is Board(Color)

Syntax

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
validateMoves	BoardMoves	$\mathbb{B}$	
updateBoard	BoardMoves		

Semantics

Access Routine Semantics

validateMoves(b):

- output : out := $|b| \geq 1 \wedge \forall(p : PointT | p \in b : validPoint(p)) \wedge validPath(b)$
- 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())
validPath: BoardMoves $\rightarrow \mathbb{B}$
validPath(b) $\equiv \forall(i : \mathbb{N} | i \in [0..|b| - 1] : isAdjacent(i, i + 1))$

isAdjacent: $BoardMoves \times \mathbb{N} \times \mathbb{N} \to \mathbb{B}$
$\text{isAdjacent}(b, i, j) \equiv b[i].row() = b[j].row() \wedge b[i].col() = b[j].col() + 1$
$\vee\ b[i].row() = b[j].row() \wedge b[i].col() = b[j].col() - 1$
$\vee\ b[i].row() = b[j].row() - 1 \wedge b[i].col() = b[j].col()$
$\vee\ b[i].row() = b[j].row() + 1 \wedge b[i].col() = b[j].col()$

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 $\mathbb{Z} \mid x \in s : \forall(y : \mathbb{Z} | y \in x : M \geq y)) \wedge (\exists i : \mathbb{N} | validRow(i) : M \in s[i])$
- 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: $\mathbb{N} \to \mathbb{B}$
$\text{validCol}(c) \equiv (c \geq 0) \wedge (c < \text{nCol})$

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

Critique of Design

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(j)){
//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.

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 made 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. Keeping this in mind, It is rarely possible to achieve a module interface that simultaneously meets the criteria of being essential, minimal and general. This is because a general module will need to offer different functionality and be flexible (work with different types, work with different input formats (CSV vs JSON file) etc.). To offer these different features there will need to be several methods/routines that are almost similar but differ slightly (e.g parsing a JSON file vs a CSV file). In order to provide a client different functionality and make the module more flexible often there will need to be an excess number of state transitions. For example, a client may want to return the results of reading a JSON file as a single string or as a Dictionary of Key, value pairs, to offer this extra flexibility and provide both we need to have state transitions for both of these and a result the module may not always be minimal. Lastly, due to this addition of several methods to achieve and provide a range of features the module will usually not always be essential either. There may be methods that can be removed and the specification is still matched but having

them present possbily improves the code quality of the module.