Assignment 4, Specification

SFWR ENG 2AA4

April 9, 2018

CardTypes Module

Module

CardTypes

Uses

N/A

Syntax

Exported Constants

None

Exported Types

ValueT = {A, two, three, four, five, six, seven, eight, nine, ten, J, Q, K}
SuitT = {Heart, Diamond, Club, Spade}
PileType = {Tableau, Cell, Foundation}

Exported Access Programs

None

Semantics

State Variables

None

State Invariant

None

CardADT Module

Template Module

CardT

Uses

N/A

Syntax

Exported Types

CardT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
CardT	$\mathbb{Z}$, $\mathbb{Z}$	CardT	
Value		ValueT	
suit		SuitT	

Semantics

State Variables

val: ValueT
suit: SuitT

State Invariant

None

Assumptions

The constructor CardT 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. Also assume that the user will only call the constructor with real life card values and suits(i.e. A - King and Heart, Diamond, Club or Spades).

Access Routine Semantics

CardT(V, S):

- transition: $val, suit := V, S$
- output: $out := self$
- exception: None

Value():

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

Suit():

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

PileADT Module

Template Module

PileT

Uses

CardT, CardTypes

Syntax

Exported Types

PileT = ?

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
PileT		PileT	
getCard	$\mathbb{N}$	CardT	pile_empty
top		CardT	pile_empty
NumCards		$\mathbb{N}$	
addCard	CardT		
removeTop			pile_empty

Semantics

State Variables

$Cards$: std::vector<CardT>
h: $\mathbb{N}$

State Invariant

None

Assumptions

The constructor PileT 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. The user will only be using the class' methods with realistic cards, which are valid cards in all operations.

Access Routine Semantics

PileT():

- transition: $h := 0$
- output: $out := self$
- exception: none

getCard(i):

- output: $out := Cards[i]$
- exception: exc := ($|Cards| = 0 \implies$ pile_empty)

top():

- output: $out := Cards[|Cards| - 1]$
- exception: exc := ($|Cards| = 0 \implies$ pile_empty)

NumCards():

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

addCard(CardT card):

- transition: $Cards := Cards || < card >$
- exception: None

removeTop():

- transition: $Cards := Cards[0..|Cards| - 2]$ (becomes the same vector minus the last entry)
- exception: exc := ($|Cards| = 0 \implies$ pile_empty)

BoardADT Module

Template Module

BoardT

Uses

PileT, CardT, CardTypes

Syntax

Exported Types

BoardT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
BoardT	PileT	BoardT	
getPile	PileType, $\mathbb{Z}$	PileT	invalid_argument
MoveCard	PileType, $\mathbb{Z}$, PileType, $\mathbb{Z}$		inval_arg, illegal_move, fromPile_empty
Complete		$\mathbb{B}$	

Semantics

State Variables

$Tableaus$: sequence of PileT
$Cells$: sequence of PileT
$Foundations$: sequence of PileT

State Invariant

None

Assumptions

- The constructor BoardT 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. If no deck is given for constructor call with deafult constructor that generates a deck automatically and distributes uniformly.

Access Routine Semantics

BoardT($deck$):

- transition: $(i \in [0..52] | Tableaus[i\%8] := Tableaus[i\%8] || < deck[|deck| - i] >)$ (distributes a deck of cards evenly to each tableau)
- output: $out := self$
- exception: None

getPile($pType, i$):

- output: $out := ((pType = Tableau \implies Tableaus[i])|$
  $(pType = Cell \implies Cells[i])|$
  $(pType = Foundation \implies Foundations[i]))$

- exception: exc $:= ((\neg(pType \in PileTypes) \implies invalid\_argument)|$
  $(pType = Tableau \wedge (i < 0 \vee i >= 8) \implies invalid\_argument)|$
  $((pType = Cell \vee pType = Foundation) \wedge (i < 0 \vee i >= 4) \implies invalid\_argument))$

MoveCard(fromPile, i, ToPile, j):

- output: $transition :=$

$fromPile =$ Tableau, $ToPile =$ Tableau	moveTopTo($Tableaus[i], Tableaus[j]$)
$fromPile =$ Tableau, $ToPile =$ Cell	moveTopTo($Tableaus[i], Cells[j]$)
$fromPile =$ Tableau, $ToPile =$ Foundation	moveTopTo($Tableaus[i], Foundations[j]$)
$fromPile =$ Cell, $ToPile =$ Tableau	moveTopTo($Cells[i], Tableaus[j]$)
$fromPile =$ Cell, $ToPile =$ Cell	moveTopTo($Cells[i], Cells[j]$)
$fromPile =$ Cell, $ToPile =$ Foundation	moveTopTo($Cells[i], Foundations[j]$)
$fromPile =$ Foundation, $ToPile =$ Tableau	moveTopTo($Foundations[i], Tableaus[j]$)
$fromPile =$ Foundation, $ToPile =$ Cell	moveTopTo($Foundations[i], Cells[j]$)
$fromPile =$ Foundation, $ToPile =$ Foundation	moveTopTo($Foundations[i], Foundations[j]$)

- exception: exc :=

fromPile, ToPile	Condition	Exception
	fromPileEmpty(fromPile, i)	fromPile_empty
	¬PileTypeValid(fromPile, i) ∨ ¬PileTypeValid(ToPile, j)	invalid_argument
	¬indexValid(fromPile, i) ∨ ¬indexValid(ToPile, j)	invalid_argument
Tab, Tab	¬canMoveTabToTab(i, j)	illegal_move
Tab, Tab	¬canMoveTabToTab(i, j)	illegal_move
Tab, Cell	¬canMoveTabToCell(i, j)	illegal_move
Tab, Found	¬canMoveTabToFoundation(i, j)	illegal_move
Cell, Tab	¬canMoveCellToTab(i, j)	illegal_move
Cell, Cell	¬canMoveCellToCell(i, j)	illegal_move
Cell, Found	¬canMoveCellToFoundation(i, j)	illegal_move
Found, Tab	¬canMoveFoundationToTab(i, j)	illegal_move
Found, Cell	¬canMoveCellToCell(i, j)	illegal_move
Found, Found	¬canMoveFoundationToFoundation(i, j)	illegal_move

Complete():

- output: $out := (i \in [0..3] | Foundations[i].NumCards() = 0 \implies True)$
- exception: exc := None

Local Functions

moveTopTo: PileT x PileT → PileT x PileT
moveTopTo$(fromPile, ToPile) \equiv ToPile$.addCard($fromPile$.top()) → $fromPile$.removeTop()

Potential Moves:
fromPileEmpty: PileType x $\mathbb{Z} \rightarrow \mathbb{B}$
fromPileEmpty$(fromPile, j)$
$\equiv ((fromPile = Tableau \implies Tableaus[i].\text{NumCards}() = 0 \implies True))$

PileTypeValid: PileType x $\mathbb{Z} \rightarrow \mathbb{B}$
PileTypeValid$(fromPile, j)$
$\equiv ((fromPile = \text{Tableau} \lor fromPile = \text{Cell} \lor fromPile = \text{Foundation}))$

indexValid: PileType x $\mathbb{Z} \rightarrow \mathbb{B}$
indexValid$(fromPile, j)$
$\equiv ((fromPile = \text{Tableau} \land i < 0 \land i >= 8) \lor$
$(fromPile = \text{Cell} \land i < 0 \land i >= 4) \lor$
$(fromPile = \text{Foundation} \land i < 0 \land i >= 4) \implies True)$

canMoveTabToTab: $\mathbb{Z}$ x $\mathbb{Z} \rightarrow \mathbb{B}$
canMoveTabToTab(i, j)
$\equiv ((Tableaus[i]$.top().Value() $= Tableaus[j]$.top().Value()-1) $\wedge$
$((Tableaus[i]$.top().Suit() $=(Heart \vee Diamond) \wedge Tableaus[j]$.top().Suit() $=(Club \vee Spade)) \vee$
$(Tableaus[i]$.top().Suit() $=(Club \vee Spade) \wedge Tableaus[j]$.top().Suit() $=(Heart \vee Diamond)))$
$\implies True)$

canMoveTabToCell: $\mathbb{Z}$ x $\mathbb{Z} \rightarrow \mathbb{B}$
canMoveTabToCell(i, j)
$\equiv ((|Tableaus[i]| \neq 0 \wedge |Cells[j] = 0 \implies True)$

canMoveTabToFoundation: $\mathbb{Z}$ x $\mathbb{Z} \rightarrow \mathbb{B}$
canMoveTabToFoundation(i, j)
$\equiv ((Tableaus[i]$.top().Value() $= Foundations[j]$.top().Value()+1) $\wedge$
$(Tableaus[i]$.top().Suit() $= Foundations[j]$.top().Suit()) $\implies True)$

canMoveCellToTab: $\mathbb{Z}$ x $\mathbb{Z} \rightarrow \mathbb{B}$
canMoveTabToTab(i, j)
$\equiv ((|Cells[i]| = 1) \wedge (Cells[i]$.top().Value() $= Tableaus[j]$.top().Value()-1) $\wedge$
$((Cells[i]$.top().Suit() $=(Heart \vee Diamond) \wedge Tableaus[j]$.top().Suit() $=(Club \vee Spade)) \vee$
$(Cells[i]$.top().Suit() $=(Club \vee Spade) \wedge Tableaus[j]$.top().Suit() $=(Heart \vee Diamond)))$
$\implies True)$

canMoveCellToCell: $\mathbb{Z}$ x $\mathbb{Z} \rightarrow \mathbb{B}$
canMoveCellToCell(i, j)
$\equiv ((|Cells[i]| \neq 0 \wedge |Cells[j] = 0 \implies True)$

canMoveCellToFoundation: $\mathbb{Z}$ x $\mathbb{Z} \rightarrow \mathbb{B}$
canMoveCellToFoundation(i, j)
$\equiv ((Cells[i]$.top().Value() $= Foundations[j]$.top().Value()+1) $\wedge$
$(Cells[i]$.top().Suit() $= Foundations[j]$.top().Suit()) $\implies True)$

canMoveFoundationToTab: $\mathbb{Z}$ x $\mathbb{Z} \rightarrow \mathbb{B}$
canMoveTabToTab(i, j)
$\equiv ((|Foundations[i]| = 1) \wedge (Foundations[i]$.top().Value() $= Tableaus[j]$.top().Value()-1)
$\wedge$
$((Foundations[i]$.top().Suit() $=(Heart \vee Diamond) \wedge Tableaus[j]$.top().Suit() $=(Club \vee$
$Spade)) \vee$
$(Foundations[i]$.top().Suit() $=(Club \vee Spade) \wedge Tableaus[j]$.top().Suit() $=(Heart \vee Diamond)))$
$\implies True)$

canMoveFoundationToCell: $\mathbb{Z}$ x $\mathbb{Z}\rightarrow$ $\mathbb{B}$
canMoveCellToCell(i, j)
$\equiv ((|Foundations[i]| \neq 0 \wedge |Cells[j] = 0 \implies True)$

canMoveFoundationToFoundation: $\mathbb{Z}$ x $\mathbb{Z}\rightarrow$ $\mathbb{B}$
canMoveFoundationToFoundation(i, j)
$\equiv ((Foundations[i]$.top().Value() $= Foundations[j]$.top().Value()+1) $\wedge$
$(Foundations[i]$.top().Suit() $= Foundations[j]$.top().Suit()) $\implies True)$