Assignment 4, Part 1, Specification

Bilal Jaffry

April 9, 2018

The purpose of this software design exercise is to design and implement a functional version of the solitaire card game played using a deck of cards. This documentation shows the complete specification for implementation and subsequent testing.

Card ADT Module

Module

Card

Uses

N/A

Syntax

Exported Constants

numOfRanks = 13
numOfSuits = 4

Exported Types

Card = ?

RANK = {ACE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, JACK, QUEEN, KING}

SUIT = {SPADE, HEART, CLUB, DIAMOND}

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
Card	SUIT, RANK	Card	
getRank		RANK	
getSuit		SUIT	
setRank	RANK		
setSuit	SUIT		

Semantics

State Variables

st : SUIT
rnk : RANK

State Invariant

None

Assumptions

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

Access Routine Semantics

Card(s, r):

- transition: $st, rnk := s, r$
- output: $out := self$
- exception: None

getRank():

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

getSuit():

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

setRank(r):

- transition: $rnk := r$
- exception: None

setSuit(s):

- transition: $st := s$
- exception: None

Deck ADT Module

Template Module

Deck

Uses

Card from CardADT.

Syntax

Exported Constants

MAX_CARDS = 52

Exported Types

Deck = ?

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
Deck		Deck	
dealCard		Card	outside_bounds
shuffleDeck			

Semantics

State Variables

$numCard$: $\mathbb{Z}$
$deckOfCards$: seq of Card

State Invariant

None

Assumptions

The constructor Deck 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

Deck():

- transition: $\forall(i : \mathbb{N} | i \in MAX\_CARDS : deckOfCards[i] = Card)$
- output: $out := \mathit{self}$
- exception: None

dealCard():

- transition: $numCard = numCard + 1$
- output: $out := deckOfCards[numCard]$
- exception: (numCard > MAX_CARDS | numCard < 0) $\implies$ outside_bounds()

shuffleDeck():

- transition: $\forall$(i : $\mathbb{N}$| i $\in$ MAX_CARDS - 1 : swap(deckOfCards(i),deckOfCards(i+1))
- exception: None

Local Functions

swap: Card × Card → None
swap (a, b): Takes two cards and swaps them around. ≡ (a = b, b = a)

Board Module

Template Module

Board

Uses

Deck from DeckADT and Card from CardADT.

Syntax

Exported Constants

NUM_OF_FREECELLS = 4;
NUM_OF_HOMECELLS = 4;
NUM_OF_GAMECELLS = 8;

Exported Types

Board = ?

Exported Access Programs

Routine name	In	Out	Exceptions
Board		Board	
moveGameToGame	ℤ, ℤ, ℤ		invalid_move(),outside_bounds()
moveGameToHome	ℤ, ℤ		invalid_card(),invalid_move(),empty_outsid
moveGameToFree	ℤ, ℤ		invalid_move(),outside_bounds()
moveFreeToGame	ℤ, ℤ		invalid_move(),empty_outside_bounds()
moveFreeToHome	ℤ, ℤ		invalid_card(),invalid_move(),empty_cells()
WinGame		𝔹	
isEmptyGameCell	ℤ	𝔹	outside_bounds()
isEmptyHomeCell	ℤ	𝔹	outside_bounds()
isEmptyFreeCell	ℤ	𝔹	outside_bounds()
suitMatch	SUIT, SUIT	𝔹	
rankMatch	RANK, RANK	𝔹	
getGameCells		seq of (seq of Card)	empty_cells()
getFreeCells		seq of Card	empty_cells()
getHomeCells		seq of (seq of Card)	empty_cells()
setGameCells	Card, ℤ		outside_bounds()

Semantics

State Variables

$HomeCells$: seq of (seq of Card)
$FreeCells$: seq of Card
$GameCells$: seq of (seq of Card)
dck: Deck
win: $\mathbb{B}$
$emptyFreeCell$: seq of $\mathbb{B}$

State Invariant

None

Assumptions

The constructor Board 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

Board():

- transition: $|HomeCells|, |GameCells|, |FreeCells|, win :=$ NUM_OF_HOMECELLS, NUM_OF_FREECELLS, NUM_OF_GAMECELLS, FALSE

  transition: dck.shuffleDeck(), $\forall(i : \mathbb{N}|i \in$ MAX_CARDS: $(\forall(j : \mathbb{N}|j \in$ NUM_OF_GAMECELLS: GameCells[j] = dck.dealCard())))
- output: $out := self$
- exception: None

moveGameToGame(numOfCards, col1, col2):

- transition: $\forall$ (i :$\mathbb{N}\vee$i $\in$ numOfCards: GameCells[col2][|GameCells[col2]|-1] = GameCells[col1][|GameCells[col1]|-1])
- exception: (numOfCards > availableMoves(col2)) $\vee$ ($\neg$ isEmptyGameCell(col1)) $\vee$ ($\neg$ rankMatch(GameCells[col1][|GameCells[col1]|-1].getRank(), GameCells[col2][|GameCells[col2]|-1].getRank())) $\vee$ ($\neg$ suitMatch(GameCells[col2][|GameCells[col2]|-1].getSuit(),(GameCells[col1][|GameCells[col1]|-1].getSuit())) $\implies$ invalid_move()

  exception: ($\neg$ validCol(col1) $\vee$ $\neg$ validCol(col2)) $\implies$ outside_bounds()

moveGameToHome(gameCol, homeCol):

- transition: HomeCells[homeCol][|HomeCells[homeCol]| $-$ 1] = GameCells[gameCol][|GameCells[gameCol]| $-$ 1]
- exception: $\neg$(isEmptyGameCell(gameCol) $\wedge$ validGameCol(gameCol) $\wedge$ validHomeCol(homeCol)) $\implies$ empty_outside_bounds()

exception:
$\neg$ rankMatch(GameCells[gameCol][|GameCells[gameCol]| $-$ 1].getRank(),
HomeCells[homeCol][|HomeCells[homeCol]| $-$ 1].getRank())
$\vee$
$\neg$ suitMatch(GameCells[gameCol][|GameCells[gameCol]| $-$ 1].getSuit(),
HomeCells[homeCol][|HomeCells[homeCol]| $-$ 1].getSuit()) $\implies$ invalid_move()

moveGameToFree(gameCol, freeCol):

- transition:

  FreeCells[freeCol], emptyFreeCell[freeCol] :=
  GameCells[gameCol][|GameCells[gameCol]| $-$ 1], emptyFreeCell[freeCol] = FALSE

- exception: $\neg$ emptyFreeCell[freeCol] $\vee$ isEmptyGameCell(gameCol) $\implies$ invalid_move()

moveFreeToGame(freeCol, gameCol):

- transition:

  GameCells[gameCol][|GameCells[gameCol]| - 1], emptyFreeCell[freeCol] :=
  FreeCells[freeCol], TRUE

- exception:
  $\neg$ rankMatch(GameCells[gameCol][|GameCells[gameCol]|-1].getRank(),
  FreeCells[freeCol].getRank())
  $\vee$
  $\neg$ suitMatch(FreeCells[freeCol].getSuit(),
  GameCells[gameCol][|GameCells[gameCol]| - 1].getSuit()) $\implies$ invalid_move()

  exception: isEmptyFreeCell(freeCol) $\vee$ $\neg$ validGameCol(gameCol) $\vee$ $\neg$ validFreeCol(freeCol)
  $\implies$ empty_outside_cells()

moveFreeToHome(freeCol, homeCol):

- transition:

  HomeCells[homeCol][|HomeCells[homeCol]| - 1], emptyFreeCell[freeCol] :=
  FreeCells[freeCol], emptyFreeCell[freeCol] = TRUE

- exception: isEmptyFreeCell(gameCol) $\implies$ empty_cells()

  exception:
  $\neg$ rankMatch(FreeCells[freeCol].getRank(),
  HomeCells[homeCol][|HomeCells[homeCol]|-1].getRank())
  $\vee$
  $\neg$ suitMatch(FreeCells[freeCol].getSuit(),
  HomeCells[homeCol][|HomeCells[homeCol]| - 1].getSuit()) $\implies$ invalid_move()

isEmptyGameCell(col):

- output: *out* := GameCells[col].empty()
- exception: $\neg$ ($0 \leq$ col $<$ NUM_OF_GAMECELLS) $\implies$ outside_bounds()

isEmptyHomeCell(col):

- output: *out* := HomeCells[col].empty()
- exception: $\neg$ ($0 \leq$ col $<$ NUM_OF_HOMECELLS) $\implies$ outside_bounds()

isEmptyFreeCell(col):

- output: *out* := emptyFreeCell[col]
- exception: $\neg$ ($0 \leq$ col $<$ NUM_OF_FREECELLS) $\implies$ outside_bounds()

getGameCells():

- output: *out* := GameCells
- exception: GameCells.empty() $\implies$ empty_cells()

getHomeCells():

- output: *out* := HomeCells
- exception: HomeCells.empty() $\implies$ empty_cells()

getFreeCells():

- output: *out* := FreeCells

- exception: FreeCells.empty() $\implies$ empty_cells()

setGameCells(a, i):

- transition: GameCells[i]|| $\langle$ a $\rangle$
- exception: $\neg$ validGameCol(i) $\implies$ outside_bounds()

rankMatch(a, b):

- output: a $-$ b $\equiv$ 1
- exception: None

suitMatch(a, b):

- output: (a + 3 $\equiv$ b) $\vee$ (a $-$ 3 $\equiv$ b) $\vee$ (a + 1 $\equiv$ b) $\vee$ (a $-$1 $\equiv$ b)
- exception: None

WinGame():

- output: +(i : $\mathbb{N}$| i $\in$ NUM_OF_HOMECELLS $\wedge$ |HomeCells[i]| $\equiv$ numOfRanks: 1) $\equiv$ NUM_OF_HOMECELLS
- exception: None

Local Functions

validGameCol: $\mathbb{Z} \rightarrow \mathbb{B}$
validGameCol($gameCol$) $\equiv 0 \leq gameCol \leq$ (NUM_OF_GAMECELLS)

validFreeCol: $\mathbb{Z} \rightarrow \mathbb{B}$
validFreeCol($freeCol$) $\equiv 0 \leq freeCol \leq$ (NUM_OF_FREECELLS)

validGameCol: $\mathbb{Z} \rightarrow \mathbb{B}$
validHomeCol($homeCol$) $\equiv 0 \leq homeCol \leq$ (NUM_OF_HOMECELLS)

availableMoves $\mathbb{Z} \rightarrow \mathbb{Z}$
availableMoves($targetCol$) $\equiv$

freeSpots = +(i : $\mathbb{N}$|i $\in$ NUM_OF_FREECELLS $\wedge$ isEmptyFreeCell(i) : 1)
gameSpots = +(j : $\mathbb{N}$|j $\in$ NUM_OF_GAMECELLS $\wedge$ isEmptyGameCell(j) : 1)
$\vee$ $-$(k : $\mathbb{N}$|k = targetCol : $-$1)

$\implies$ availableMoves = freeSpots $\times$ $2^{gameSpots}$