Assignment 5, Part 1, Specification

SFWR ENG 2AA4

April 9, 2018

The purpose of this software design exercise is to design, specify, implement and test a module for storing the state of an Freecell game.

CardADT Module

Module

CardT

Uses

N/A

Syntax

Exported Constants

Size = 52

Exported Types

CardT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
CardT	string, string	CardT	
Suit		string	
Face		string	

Semantics

State Variables

$face$: string
$suit$: string

State Invariant

None

Assumptions

The CardT method is called for the abstract object before any other access routine is called for that object. The CardT method can be used to return the state of the game to the state of a new game.

Access Routine Semantics

CardT(a, b):

- transition: face, suit = a,b
- output: *out* := *self*
- exception : none

Suit():

- output: *out* := suit
- exception : none

Face():

- output: *out* := face
- exception : none

DeckOfCardsADT Module

Module

DeckOfCardsT

Uses

CardT

Syntax

Exported Constants

Size = 52

Exported Types

DeckofCardsT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
DeckOfCardsT		CardT	
getColor	CardT	string	
shuffle		CardT	
dealCard		CardT	

Semantics

State Variables

$deck : cardT$

State Invariant

None

Assumptions

The DeckOfCardsT method is called for the abstract object before any other access routine is called for that object. The DeckOfCardsT method can be used to return the state of the game to the state of a new game.

Access Routine Semantics

DeckOfCardsT():

- transition: string faces[ ] = {”Ace”, ”2”, ”3”, ”4”, ”5”, ”6”, ”7”, ”8”, ”9”, ”10”, ”Jack”, ”Queen”, ”King”}
  string suits[ ] = {”Hearts”, ”Diamonds”, ”Clubs”, ”Spades”}
- output: *out* := *deck*
- exception : none

getColor(a):

- output: *out* := color
- exception : none

shuffle():

- output: *out* := face, suit
- exception None

dealCard():

- output: *out* := *deck*
- exception None

BoardADT Module

Module

BoardT

Uses

CardT, DeckOfCardsT

Syntax

Exported Constants

Size = 52

Exported Types

BoardT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
BoardT		CardT, BoardT, BoardT	
get	CardT	BoardT	
move	CardT,CardT	BoardT	
put	CardT	BoardT	
put	CardT,CardT	BoardT	
check	CardT	BoardT	

Semantics

State Variables

$s : CardT$
$a : BoardT$
$b : BoardT$

State Invariant

None

Assumptions

The BoardT method is called for the abstract object before any other access routine is called for that object. The BoardT method can be used to return the state of the game to the state of a new game.

Access Routine Semantics

BoardT():

- transition:
  s := CardT S[7][8]
  a := BoardT D[1][4]
  b := BoardT D[1][4]
- output: *out* := *self*
- exception : none

get(c):

- transition:
  s[0][0]:= c
- output: *out* := a
- exception:
  s[i][j] $\neq$ null $\Rightarrow$ invalid_argument

move(a, b):

- transition: (getColor(a)$\neq$getColor(b) $\wedge$ $|a.Face() - b.Face()| = 1$) $\Rightarrow$ put(a, b)
- output: *out* := s, a, b
- exception: none

put(a, b):

- transition: s[i][j] = a, s[i+1][j] = b
- output: *out* := s, a, b
- exception: (i = 6) $\Rightarrow$ Invalid_moving

put(a):

- transition: $(a.Face() \neq Ace \wedge b[0][j] = null) \Rightarrow$ j ∈ {0..3} b[0][j] = a
  $(a.Face() = Ace \wedge b[0][j] = null) \Rightarrow$ j ∈ {0..3} a[0][j] = a

- output: $out := b, a$

- exception: none

check(a):

- transition: $(a.Suit() = b[i][j].Suit() \wedge |a.Face() - b[i][j].Face()| = 1) \Rightarrow$ b[i+1][j]=a

- output: $out := b$

- exception: none