# Assignment 4, Specification

SFWR ENG 2AA4 - Immanuel Odisho
400074199

April 7, 2018

The purpose of this software design is to store the state of a game of Freecell. This document shows the specification for the design.[1]

[1]"This specification file used SFWRENG 2AA4 A3 2018 specifications as a template"

# Card Types Module

## Module

CardTypes

## Uses

N/A

## Syntax

### Exported Constants

None

### Exported Types

SuitT = {club,diamond,heart,spade}
RankT = {ace,two,three,four,five,six,seven,eight,nine,ten,jack,queen,king}
ColourT = {black,red}

### Exported Access Programs

None

## Semantics

### State Variables

None

### State Invariant

None

# CardADT Module

## Template Module

CardT

## Uses

CardTypes

## Syntax

### Exported Types

CardT = ?

### Exported Access Programs

| Routine name | In | Out | Exceptions |
|---|---|---|---|
| CardT | RankT, SuitT | CardT | |
| getRank | | RankT | |
| getSuit | | SuitT | |
| getColour | | ColourT | |

## Semantics

### State Variables

$rank$: RankT
$suit$: SuitT
$colour$: ColourT

### 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.

**Access Routine Semantics**

CardT($r, s$):

- transition: $rank, suit, colour := r, s, suit = spade \Rightarrow black | suit = club \Rightarrow black | True \Rightarrow red$
- output: $out := self$
- exception: None

getRank():

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

getSuit():

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

getColour():

- output: $out := colour$
- exception: noindent

# Deck ADT Module

## Template Module

DeckT

## Uses

CardTypes
CardT

## Syntax

### Exported Types

CardT = ?

### Exported Access Programs

| Routine name | In | Out | Exceptions |
|---|---|---|---|
| DeckT | | DeckT | |
| getDeck | | seq of CardT | |
| getRandDeck | | seq of CardT | |
| size | | $\mathbb{N}$ | |

## Semantics

### State Variables

*deck*: seq of CardT

### State Invariant

None

### Assumptions

The constructor DeckT 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**

DeckT():

- transition: $\forall(r : RankT | r \in RankT : \forall(s : SuitT | s \in SuitT : deck || < CardT(r, s) > ))$
- output: $out := self$
- exception: None

getDeck():

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

getRandDeck():

- transition: $\forall(i : \mathbb{N} | i \in [0..99] : swap(randInt(), randInt())$
- output: $out := deck$
- exception: None

size():

- output: $out := |deck|$
- exception: noindent

## Local Functions

swap: $\mathbb{N} \times \mathbb{N} \to NULL$
swap(a,b)
$\equiv$ deck[a],deck[b] := deck[b],deck[a]

randInt: $NULL \to \mathbb{N}$
randInt()
$\equiv$ return a random integer within the range [0..51]

# Game State Module

## Template Module

GameState

## Uses

CardTypes
CardTypes
DeckT

## Syntax

### Exported Types

GameState = ?

**Exported Access Programs** [2]

| **Routine name** | **In** | **Out** | **Exceptions** |
|---|---|---|---|
| GameState | DeckT | GameState | wrong_size_Deck |
| cardInTab | $\mathbb{Z}$, CardT | $\mathbb{B}$ | invalid_argument |
| numEmptySpots | | $\mathbb{N}$ | |
| TabtoFree | $\mathbb{Z}$, $\mathbb{Z}$ | | invalid_argument<br>invalid_move<br>full_cells |
| FreetoTab | $\mathbb{Z}$, $\mathbb{Z}$ | | invalid_argument<br>invalid_move |
| TabtoFound | $\mathbb{Z}$, $\mathbb{Z}$ | | invalid_argument<br>invalid_move |
| FreetoFound | $\mathbb{Z}$, $\mathbb{Z}$ | | invalid_argument<br>invalid_move |
| TabtoTab | $\mathbb{Z}$, $\mathbb{Z}$ | | invalid_argument<br>invalid_move |
| getFreeCell | $\mathbb{Z}$ | CardT | invalid_argument<br>empty_cell |
| viewTab | $\mathbb{Z}$ | seq of CardT | invalid_argument<br>empty_cell |
| getTopFound | $\mathbb{Z}$ | CardT | invalid_argument<br>empty_cell |
| getTopTab | $\mathbb{Z}$ | CardT | invalid_argument<br>empty_cell |
| validMovesRem | | $\mathbb{B}$ | |
| winCondition | | $\mathbb{B}$ | |

## Semantics

### State Variables

$cards$: seq of CardT
$tableaus$: seq of (seq of CardT)
$freecells$: seq of (seq of CardT)
$foundations$: seq of (seq of CardT)

[2]although the methods are not as general as they could be, this was intentionally done so that the interface would be much more easier to use and increase useability, the same can be said about minimalism

#### State Invariant

$|freecells| = 4$
$\forall(i : \mathbb{N}|i \in [0..3] : |freecells[i]| \leq 1)$
$|tableaus| = 8$
$|foundations| = 4$

#### Assumptions

The constructor GameState 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

GameState(deck):

- transition: cards := deck
- output: *out* := *self*
- exception: None

cardInTab(t,c):

- output: $out := \exists(card : CardT|card \in tableaus[t] : c.getRank() = card.getRank() \wedge c.getSuit() = card.getSuit())$
- exception: $\neg(0 \leq t \leq 7) \Rightarrow invalid\_argument$

numEmptySpots():

- output: $out := (+(i : \mathbb{N}|i \in [0..|freecells| - 1] : |freecells[i]| = 0 \Rightarrow 1)) + (+(i : \mathbb{N}|i \in [0..|tableaus| - 1] : |tableaus[i]| = 0 \Rightarrow 1))$
- exception: None

TabtoFree(t,f):

- transition: $freecells[f], tableaus[t] := freecells[f] \mid\mid tableaus[t][|tableaus[t]| - 1]$ , $tableaus[t][0..|tableaus[t]| - 2]$
- exception: $|freecells[f]|! = 0 \Rightarrow invalid\_move \mid |tableaus[t]| = 0 \Rightarrow invalid\_move \mid \neg(0 \leq t \leq 7) \Rightarrow invalid\_argument \mid \neg(0 \leq f \leq 3) \Rightarrow invalid\_argument \mid \neg((\exists(i : \mathbb{N}|i \in [0..3] : |freecells| = 0))) \Rightarrow full\_cells$

FreetoTab(f,t):

- transition: $tableaus[t], freecells[f] := tableaus[t] \mid\mid freecells[f][0]\ ,\ freecells[f][0:0]$
- exception: $\neg(validMovefT(f,t) \Rightarrow invalid\_move \mid \neg(0 \leq t \leq 7) \Rightarrow invalid\_argument \mid \neg(0 \leq f \leq 3) \Rightarrow invalid\_argument$

TabtoFound(t,F):

- transition: $foundations[F], tableaus[t] := foundations[F] \mid\mid tableaus[t][|tableaus[t]|-1]\ ,\ tableaus[t][|tableaus[t]|-2]$
- exception: $\neg(validMovetF(t,F) \Rightarrow invalid\_move \mid \neg(0 \leq F \leq 3) \Rightarrow invalid\_argument \mid \neg(0 \leq t \leq 7) \Rightarrow invalid\_argument$

FreetoFound(f,F):

- transition: $foundations[F], freecells[f] := foundations[F] \mid\mid freecells[f][0]\ ,\ freecells[f][0:0]$
- exception: $\neg(validMovefF(f,F) \Rightarrow invalid\_move \mid \neg(0 \leq F \leq 3) \Rightarrow invalid\_argument \mid \neg(0 \leq f \leq 3) \Rightarrow invalid\_argument$

TabtoTab(t,T):

- transition: $tableaus[T], tableaus[t] := tableaus[T] \mid\mid tableaus[t][|tableaus[t]|-1]\ ,\ tableaus[t][|tableaus[t]|-1]$
- exception: $\neg(validMovetT(t,T) \Rightarrow invalid\_move \mid \neg(0 \leq t \leq 7) \Rightarrow invalid\_argument \mid \neg(0 \leq T \leq 7) \Rightarrow invalid\_argument$

getFreecell(i):

- output: $out := freecells[i][0]$
- exception: $\neg(0 \leq i \leq 3) \Rightarrow invalid\_argument \mid |freecells[i]| = 0 \Rightarrow empty\_cell$

viewTab(i):

- output: $out := tableaus[i]$
- exception: $\neg(0 \leq i \leq 7) \Rightarrow invalid\_argument \mid |tableaus[i]| = 0 \Rightarrow empty\_cell$

getTopFound(i):

- output: $out := foundations[i][|foundations[i]| - 1]$
- exception: $\neg(0 \leq i \leq 3) \Rightarrow invalid\_argument \mid |foundations[i]| = 0 \Rightarrow empty\_cell$

getTopTab(i):

- output: $out := tableaus[i][|tableaus[i]| - 1]$
- exception: None

validMovesRem():

- output: $out := (\exists(i, j : \mathbb{N}|i \in [0..7], j \in [0..3] : validMovetF(i, j))) \vee (\exists(i, j : \mathbb{N}|i \in [0..3], j \in [0..3] : validMovefF(i, j))) \vee (\exists(i, j : \mathbb{N}|i \in [0..3], j \in [0..7] : validMovefT(i, j))) \vee (\exists(i, j : \mathbb{N}|i \in [0..7], j \in [0..7] : validMovetT(i, j)))$
- exception: None

winCondition():

- output: $\forall(i : \mathbb{N}|i \in [0..3] : (inOrder(foundation[i]))$
- exception: $\neg(0 \leq i \leq 7) \Rightarrow invalid\_argument \mid |tableaus[i]| = 0 \Rightarrow empty\_cell$

## Local Functions

validMovefF: $\mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{B}$
validMovefF(f,F)

$\equiv |freecells[f]| = 0 \Rightarrow False \mid$
$(|foundations[F]| = 0 \wedge freecells[f][0].getRank()! = ace \Rightarrow False) \mid$
$\neg((freecells[f][0].getSuit() = foundations[F][|foundations[F]| - 1].getSuit()) \wedge$
$(getVal(freecells[f][0]) + 1 = getVal(foundations[F][|foundations[F]| - 1]) \Rightarrow False \mid$
True

validMovefT: $\mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{B}$
validMovefT(f,t)

$\equiv |freecells[f]| = 0 \Rightarrow False \mid$
$(|tableaus[t]| = 0 \Rightarrow True)$
$\neg(\neg(freecells[f][0].getColour() = tableaus[F][|tableaus[F]| - 1].getColour()) \wedge$
$(getVal(freecells[f][0]) - 1 = getVal(tableaus[F][|tableaus[F]| - 1]))) \Rightarrow False \mid$ True

validMovetF: $\mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{B}$
validMovetF(t,F)

$\equiv |tableaus[f]| = 0 \Rightarrow False$ |
$(|foundations[F]| = 0 \wedge tableaus[f][|tableaus[f]| - 1].getRank()! = ace \Rightarrow False)$ |
$\neg((tableaus[f][|tableaus[f]|-1].getSuit() = foundations[F][|foundations[F]|-1].getSuit()) \wedge (getVal(tableaus[f][|tableaus[f]| - 1]) + 1 = getVal(foundations[F][|foundations[F]| - 1]))) \Rightarrow False$ |
True

validMovetT: $\mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{B}$
validMovetT(t,T)

$\equiv |tableaus[t]| = 0 \Rightarrow False$ | $(|tableaus[T]| = 0 \Rightarrow True)$ |
$\neg(\neg(tableaus[t][|tableaus[t]|-1].getColour() = tableaus[t][|tableaus[t]|-1].getColour()) \wedge (getVal(tableaus[t][|tableaus[t]| - 1]) - 1 = getVal(tableaus[T][|tableaus[T]| - 1]))) \Rightarrow False$ | True

inOrder: seq of CardT $\rightarrow \mathbb{B}$
inOrder(cards)

$\equiv \neg(|cards| = 13) \Rightarrow False \mid \forall(i : \mathbb{N} | i \in [0..11] : (cards[i].getSuit() = cards[i + 1].getSuit()) \wedge (getVal(cards[i]) + 1 = getVal(cards[i + 1])))$

getVal: CardT $\rightarrow \mathbb{N}$
getVal(c) $\equiv$

| | |
|---|---|
| $c.getRank() = ace$ | 1 |
| $c.getRank() = two$ | 2 |
| $c.getRank() = three$ | 3 |
| $c.getRank() = four$ | 4 |
| $c.getRank() = five$ | 5 |
| $c.getRank() = six$ | 6 |
| $c.getRank() = seven$ | 7 |
| $c.getRank() = eight$ | 8 |
| $c.getRank() = nine$ | 9 |
| $c.getRank() = ten$ | 10 |
| $c.getRank() = jack$ | 11 |
| $c.getRank() = queen$ | 12 |
| $c.getRank() = king$ | 13 |