Pyramid

December 27, 2016

Contents

1 Card

```
class Card

instance variables

  private value: nat;
  private left: [Card];
    private right: [Card];

operations

  --Card Constructor ex: create p := new Card(1,nil,nil);
  public Card: int * [Card] * [Card] ==> Card
  Card(v,l,r) ==
  (
   value := v;
   left := l;
   right := r;
   return self
  )
  post value = v and left = l and right = r;



  --Choose this card, get its value ex: print p.chooseCard()
  public chooseCard: () ==> nat
  chooseCard() ==
  (
   return value;
  );

  -- Set a left child card
  public setLeft: Card ==> ()
  setLeft(l) ==
  (
   left := l;
   return;
  );
  -- Set a right child card
  public setRight: Card ==> ()
  setRight(r) ==
  (
   right := r;
   return;
  );



functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end Card
```

2 Game

```
class Game
-- Pyramid has 28 cards.
instance variables
  private deck: seq of nat;
  -- The Stock: The facedown pile on the bottom left. It is used to draw cards from and put on
      the Waste.
  private stock: seq of nat;
  -- The Waste: The faceup pile next to the Stock. Cards on the Waste can be matched to cards in
      the Pyramid to get rid of the Pyramid cards.

  private waste: seq of nat;
  -- private cardDeck: seq of Card;
  inv len deck = 52;
operations
    -- create g := new Game();
  public Game: () ==> Game
  Game() ==
  (
   deck := [1,2,3,4,5,6,7,8,9,10,
   11,12,13,14,15,16,17,18,19,20,
   21,22,23,24,25,26,27,28,29,30,
   31,32,33,34,35,36,37,38,39,40,
   41,42,43,44,45,46,47,48,49,50,

   51,52];
   shuffle();

   -- cardDeck := [];

     -- for counter = 1 to len deck do
     -- (
     --  dcl c : Card;
     -- c := new Card(counter,nil,nil);
     --  cardDeck := cardDeck ^ [c];
   -- );
   IO`print("Game started.\n");
   IO`print("Deck shuffled.\n");
   return self;

  );
```

```
    -- print g.shuffle();
    public shuffle: () ==> ()
    shuffle() == (
     dcl index  : nat;
     dcl x : nat;
     for counter = 1 to len deck do
     (
       index  := MATH`rand(52) + 1;
       if index = 53 then index := 52;
       x := deck (counter);
       deck (counter) := deck (index);
       deck (index) := x;

   );
    );

    -- TODO: op that draws 1 card from waste into the stock.

    -- TODO: build Pyramid function (binary tree).

    -- TODO: function that picks two cards (one of them may be the top card on the waste).


functions

traces
-- TODO Define Combinatorial Test Traces here
end Game
```

3 Pyramid

```
class Pyramid
instance variables
        private val: int;
                left: [Pyramid];
                right: [Pyramid];
types
-- TODO Define types here
values
-- TODO Define values here
operations

-- Pyramid Constructor
-- create p := new Pyramid(2,nil,nil)

public Pyramid: int * [Pyramid] * [Pyramid] ==> Pyramid
Pyramid(v,l,r) ==
(val := v; left := l; right := r; return self)
post val = v and left = l and right = r;


functions
-- TODO Define functiones here

public getValue: Pyramid -> int
getValue(p) == p.val
```

```
traces
-- TODO Define Combinatorial Test Traces here
end Pyramid
```