Stack ADT using a model SEQ[G]

Class SEQ[G] is part of library **mathmodels**.

In OOSC2, the ADT specification of a stack is given as follows:

ADT specification of stacks

TYPES

- *STACK* [*G*]

FUNCTIONS

- *put*: *STACK* [*G*] × *G* → *STACK* [*G*]
- *remove*: *STACK* [*G*] ↛ *STACK* [*G*]
- *item*: *STACK* [*G*] ↛ *G*
- *empty*: *STACK* [*G*] → *BOOLEAN*
- *new*: *STACK* [*G*]

AXIOMS

For any *x*: *G*, *s*: *STACK* [*G*]

A1 • *item* (*put* (*s*, *x*)) = *x*

A2 • *remove* (*put* (*s*, *x*)) = *s*

A3 • *empty* (*new*)

A4 • **not** *empty* (*put* (*s*, *x*))

PRECONDITIONS

- *remove* (*s*: *STACK* [*G*]) **require not** *empty* (*s*)
- *item* (*s*: *STACK* [*G*]) **require not** *empty* (*s*)

We will use SEQ[G] as a mathematical model for an abstract stack class, allowing not only classical contracts but also complete contracts.

Chart View of SEQ[G]

```
class
        SEQ [G -> attached ANY]

General
        cluster: mathmodels
        description:
                "(1) Model contracts for finite sequences for elements of type G.
                (2) Queries are side effect free and return sequences via
                    deep_twin; thus inefficient (use only for contracts).
                (3) Commands change the current sequence.
                (4) A valid index is 1..count.
                (5) Array notation can be used, as well as iteration (across).
                (6) Empty sequences can be created, or creation can be from an array.
                (7) Sequences have a first item (the head), a tail and last item.
                (8) Infix notation for prepended_by(x:G):
                seq1 |< x
                (9) Infix notation for appended_by(x:G):
                seq1 |> x
                (10) For concatenation may use infix: seq1 |++| seq2
                (11) For queries, to assert that the state is not changed,
                the postcondition is
                Current ~ old Current.deep_twin"
        create: make_empty, make_from_array

Ancestors
        DEBUG_OUTPUT*
        ITERABLE* [G]

Queries
        appended alias "|->" (v: G): [like Current] attached SEQ [G]
        as_array: ARRAY [G]
        as_function: FUN [INTEGER_32, G]
        comprehension alias "|" (exp: PREDICATE [PAIR [INTEGER_32, G]]):
                                                        [like Current] attached SEQ [G]
        concatenated alias "|++|" (other: [like Current] attached SEQ [G]):
                                                        [like Current] attached SEQ [G]
        count alias "#": INTEGER_32
        debug_output: STRING_8
        first: G
        front: [like Current] attached SEQ [G]
        has (v: G): BOOLEAN
        hold_count (exp: PREDICATE [PAIR [INTEGER_32, G]]): INTEGER_32
        inserted (v: G; i: INTEGER_32): [like Current] attached SEQ [G]
        is_contiguous_subseq_of (other: [like Current] attached SEQ [G]): BOOLEAN
        is_empty: BOOLEAN
        is_equal (other: [like Current] attached SEQ [G]): BOOLEAN
        is_subsequence_of alias "|<:" (other: [like Current] attached SEQ [G]): BOOLEAN
        item alias "[]" (i: INTEGER_32): G
        last: G
        lower: INTEGER_32
        new_cursor: ITERATION_CURSOR [G]
        out: STRING_8
        overriden (v: G; i: INTEGER_32): [like Current] attached SEQ [G]
        prepended alias "|<-" (v: G): [like Current] attached SEQ [G]
        removed (i: INTEGER_32): [like Current] attached SEQ [G]
        reversed: [like Current] attached SEQ [G]
        slice (a_start, a_end, a_step: INTEGER_32): [like Current] attached SEQ [G]
        subsequenced (i, j: INTEGER_32): [like Current] attached SEQ [G]
        tail: [like Current] attached SEQ [G]
        twin2: [like Current] attached SEQ [G]
        upper: INTEGER_32
        valid_position (pos: INTEGER_32): BOOLEAN

Commands
        append (v: G)
        concatenate (other: [like Current] attached SEQ [G])
        insert (v: G; i: INTEGER_32)
        make_empty
```

```
        make_from_array (a: ARRAY [G])
        override (v: G; i: INTEGER_32)
        prepend (v: G)
        remove (i: INTEGER_32)
        reverse
        subsequence (i, j: INTEGER_32)

Constraints
        value semantics
        value semantics
        value semantics
        value semantics
        properties:
           not is_empty implies
           Current ~ tail.prepended (first) and Current ~ front.appended (last)
```

ADT-STACK

```
note
        description: "[
                Abstract Data Type for a Stack, with value semantics.
                Classic contracts vs. Complete contracts with a model.
        ]"
        author: "JSO"

deferred class
        ADT_STACK [G -> attached ANY]

inherit
        ANY
                undefine
                        is_equal
                end

feature -- model

        model: SEQ [G]
                        -- abstract mathematical description of stack
                        -- abstraction function
                deferred
                end

feature -- queries

        count: INTEGER_32
                        -- number of items in stack
                deferred
                ensure
                        complete: Result = model.count
                end

        is_empty: BOOLEAN
                        -- is the queue empty?
                deferred
                ensure
                        complete: Result = model.is_empty
                end

        item: G
                        -- Top element of stack ("peek")
                require
                                not is_empty
                deferred
                ensure
                        complete: Result ~ model.last
                end
```

```
	is_equal (other: like Current): BOOLEAN
			-- Is `other' attached to an object considered
			-- equal to current object?
		deferred
		ensure then
			complete: model ~ other.model
		end

feature -- commands

	make_empty
			-- Initialization for `Current'.
		deferred
		ensure
			classic: is_empty
		end

	put (x: G)
			-- push 'x' on top of stack ("push")
		deferred
		ensure
			classic: count = old count + 1
			classic: item ~ x
			complete: model ~ ((old model) |-> x)
		end

	remove
			-- pop off top of stack ("pop")
		require
				not is_empty
		deferred
		ensure
			classic: count = old count - 1
			complete: (old model) ~ (model |-> (old item))
		end

feature -- axioms

	axiom (x: G)
		do
			Current.put (x)
			Current.remove
		ensure
				model ~ (old model)
		end

end -- class ADT_STACK
```

MY_STACK text

```
class
	MY_STACK [G -> attached ANY]

inherit
	ADT_STACK [G]

create
	make_empty

feature {ADT_STACK} -- Initialization

	imp: LINKED_LIST [G]
			-- implementation

	make_empty
			-- Initialization for `Current'.
		do
			create imp.make
			imp.compare_objects
		end

feature -- model

	model: SEQ [G]
			-- abstract mathematical description of stack
			-- abstraction function
		do
			create Result.make_empty
			across
				imp as cr
			loop
				Result.append (cr.item)
			end
		end

feature -- queries

	count: INTEGER_32
			-- number of items in stack
		do
			Result := imp.count
		end

	is_empty: BOOLEAN
			-- is the queue empty?
		do
			Result := imp.is_empty
		end

	item: G
			-- Top element of stack ("peek")
		do
			Result := imp.last
		end

	is_equal (other: like Current): BOOLEAN
			-- Is `other' attached to an object considered
			-- equal to current object?
		do
			Result := imp ~ other.imp
		end
```

```
feature -- command

        put (x: G)
                        -- push 'x' on top of stack ("push")
                do
                        imp.extend (x)
                end

        remove
                        -- pop off top of stack ("pop")
                do
                        imp.go_i_th (imp.count)
                        imp.remove
                end

end -- class MY_STACK
                        -- Generated by ISE Eiffel --
                        -- For more details: http://www.eiffel.com --
```