EECS 3311

Fall 2018

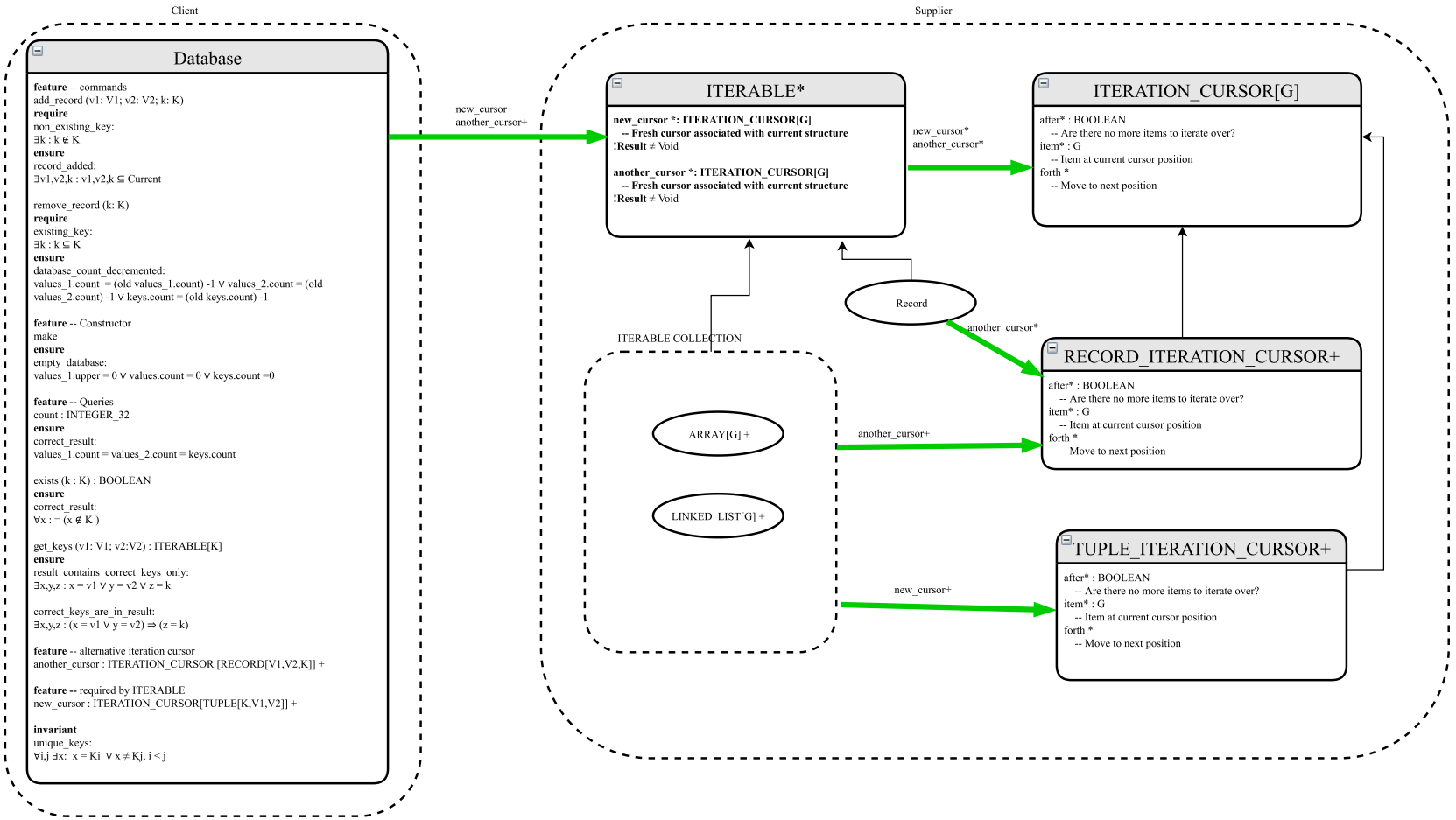
Kevin Shen

Kshen94

```
note
	description: "A DATABASE ADT mapping from keys to two kinds of values"
	author: "Kevin Shen"
	date: "$Date$"
	revision: "$Revision$"

class
	DATABASE[V1, V2, K]

inherit
	ITERABLE[TUPLE[K,V1,V2]]



create
	make

feature {EXAMPLE_DATABASE_TESTS} -- Do not modify this export status!
	-- You are required to implement all database features using these three attributes.
	keys: LINKED_LIST[K]
	values_1: ARRAY[V1]
	values_2: LINKED_LIST[V2]


feature -- feature(s) required by ITERABLE
	-- Your Task
	-- See test_iterable_databse and test_iteration_cursor in EXAMPLE_DATABASE_TESTS.
	-- As soon as you make the current class iterable,
	-- define the necessary feature(s) here.
	new_cursor: ITERATION_CURSOR[TUPLE[K,V1,V2]]
		local
			cursor : TUPLE_ITERATION_CURSOR[K,V1,V2]
		do
			create cursor.make(keys,values_1,values_2)
			Result := cursor
		end


feature -- alternative iteration cursor
	-- Your Task
	-- See test_another_cursor in EXAMPLE_DATABASE_TESTS.
	-- A feature 'another_cursor' is expected to be defined here.
	another_cursor: ITERATION_CURSOR[RECORD[V1,V2,K]]
		local
			cursor : RECORD_ITERATION_CURSOR[V1,V2,K]
		do
			create cursor.make(values_1, values_2, keys)
			result := cursor
		end

feature -- Constructor
	make
			-- Initialize an empty database.
		do
			-- Your Task
			create values_1.make_empty
			create values_2.make
			create keys.make
			keys.compare_objects
			values_1.compare_objects
			values_2.compare_objects

		ensure
			empty_database: -- Your Task
				values_1.upper = 0 and
				values_2.count = 0 and
				keys.count = 0
			-- Do not modify the following three postconditions.
			object_equality_for_keys:
				keys.object_comparison
			object_equality_for_values_1:
				values_1.object_comparison
			object_equality_for_values_2:
				values_2.object_comparison
		end

feature -- Commands

	add_record (v1: V1; v2: V2; k: K)
			-- Add a new record into current database.
		require
			non_existing_key: -- Your Task
				not(current.exists (k))
		do
			-- Your Task
			values_1.force (v1, values_1.upper +1)
			values_2.force (v2)
```

```
                keys.force (k)
        ensure
                record_added: -- Your Task
                        -- Hint: At least a record in current database.
                        -- has its key 'k', value 1 'v1', and value 2 'v2'.
                across current
                         as c
                some
                        c.item[3] ~ v2 and c.item[2] ~ v1 and c.item[1] ~ k
                end
        end

    remove_record (k: K)
                -- Remove a record from current database.s

        require
                existing_key: -- Your Task
                        current.exists (k)
        local
                new_array : ARRAY[V1]
                exit : BOOLEAN
        do
                -- Your Task
                exit := false
                across 1 |..| keys.count as c
                until exit
                loop
                        if(keys[c.item] ~ k) then
                                new_array := values_1.subarray (1, c.item -1)
                                        if (c.item /~ keys.count) then
                                                across values_1.subarray (c.item +1, values_1.count) as
v1
                                                loop
                                                        new_array.force (v1.item, new_array.count+1)
                                                end
                                        end
                                values_1 := new_array
                                keys.go_i_th (c.item)
                                keys.remove
                                values_2.go_i_th (c.item)
                                values_2.remove
                                exit := true
                        end

                end
        ensure
                database_count_decremented: -- Your Task
                        values_1.count ~ (old values_1.count) -1 and
                        values_2.count ~ (old values_2.count) -1 and
                        keys.count ~ (old keys.count) -1
                key_removed: -- Your Task
                        not(find_k_once(k))
        end

feature -- Queries

    count: INTEGER
                -- Number of records in database.
        do
                -- Your Task
                Result := values_1.count
        ensure
                correct_result: -- Your Task
                        values_1.count ~ values_2.count
                        and values_2.count ~ keys.count
        end

    exists (k: K): BOOLEAN
                -- Does key 'k' exist in the database?
        do
                -- Your Task
                result := across
                        current as c
                some
                        c.item[1] ~ k
                end
        ensure
                correct_result: -- Your Task
                        not(across
                                current as c
                        all
                                c.item[1] /~ k
                        end)
        end
```

```
	get_keys (v1: V1; v2: V2): ITERABLE[K]
			-- Keys that are associated with values 'v1' and 'v2'.
		local
			list_keys : LINKED_LIST[K]
		do
			-- Your Task
			create list_keys.make
			across
				1 |..| keys.count as c
			loop
				if (values_1[c.item] ~ v1 and values_2[c.item] ~ v2)
				then
					list_keys.sequence_put (keys[c.item])
				end
			end
			result := list_keys
		ensure
			result_contains_correct_keys_only: -- Your Task
				-- Hint: Each key in Result has its associated values 'v1' and 'v2'.
				across result as r
				all
					across current as c
					some
						c.item[2] ~ v1 and c.item[3] ~ v2 and c.item[1] ~ r.item
					end
				end

			correct_keys_are_in_result: -- Your Task
				-- Hint: Each record with values 'v1' and 'v2' has its key included in Result.
				-- Notice that Result is ITERABLE and does not support the feature 'has',
				-- Use the appropriate across expression instead.
				across 1 |..| keys.count as c
				all
					if (values_1[c.item] ~ v1 and values_2[c.item] ~ v2) then
						across  result as  r
						some
							r.item ~ keys[c.item]
						end
						else
							true
						end
					end
				end


	find_k_once(k : K): BOOLEAN
		local
			c : INTEGER
		do
			across keys as key
				loop
				if key.item ~ k then
					c := c +1
				end
			end
			if c = 1 then
				result := True
			else
				result := False
			end
		end



invariant
	unique_keys: -- Your Task
		-- Hint: No two keys are equal to each other.
		across keys as k
		all
			find_k_once(k.item)
		end

	-- Do not modify the following three class invariants.
	implementation_contraint:
		values_1.lower = 1
	consistent_keys_values_counts:
		keys.count = values_1.count
		and
		keys.count = values_2.count
	consistent_imp_adt_counts:
		keys.count = count
end
```


Client
Supplier
Database
feature -- commands
add_record (v1: V1; v2: V2; k: K)
require
non_existing_key:
∃k : k ∉ K
ensure
record_added:
∃v1,v2,k : v1,v2,k ⊆ Current
remove_record (k: K)
require
existing_key:
∃k : k ⊆ K
ensure
database_count_decremented:
values_1.count = (old values_1.count) -1 ∨ values_2.count = (old values_2.count) -1 ∨ keys.count = (old keys.count) -1
feature -- Constructor
make
ensure
empty_database:
values_1.upper = 0 ∨ values.count = 0 ∨ keys.count =0
feature -- Queries
count : INTEGER_32
ensure
correct_result:
values_1.count = values_2.count = keys.count
exists (k : K) : BOOLEAN
ensure
correct_result:
∀x : ¬ (x ∉ K)
get_keys (v1: V1; v2:V2) : ITERABLE[K]
ensure
result_contains_correct_keys_only:
∃x,y,z : x = v1 ∨ y = v2 ∨ z = k
correct_keys_are_in_result:
∃x,y,z : (x = v1 ∨ y = v2) ⇒ (z = k)
feature -- alternative iteration cursor
another_cursor : ITERATION_CURSOR [RECORD[V1,V2,K]] +
feature -- required by ITERABLE
new_cursor : ITERATION_CURSOR[TUPLE[K,V1,V2]] +
invariant
unique_keys:
∀i,j ∃x: x = Ki ∨ x ≠ Kj, i < j
new_cursor+
another_cursor+
ITERABLE*
new_cursor *: ITERATION_CURSOR[G]
-- Fresh cursor associated with current structure
!Result ≠ Void
another_cursor *: ITERATION_CURSOR[G]
-- Fresh cursor associated with current structure
!Result ≠ Void
new_cursor*
another_cursor*
ITERATION_CURSOR[G]
after* : BOOLEAN
-- Are there no more items to iterate over?
item* : G
-- Item at current cursor position
forth *
-- Move to next position
Record
another_cursor*
ITERABLE COLLECTION
ARRAY[G] +
LINKED_LIST[G] +
another_cursor+
RECORD_ITERATION_CURSOR+
after* : BOOLEAN
-- Are there no more items to iterate over?
item* : G
-- Item at current cursor position
forth *
-- Move to next position
new_cursor+
TUPLE_ITERATION_CURSOR+
after* : BOOLEAN
-- Are there no more items to iterate over?
item* : G
-- Item at current cursor position
forth *
-- Move to next position