Semester fall 2018

Fahad Qayyum

215287253

fahadq

EECS 3311 E

Lab 2

```
note
    description: "A DATABASE ADT mapping from keys to two kinds of values"
    author: "Jackie Wang and Fahad Qayyum"
    date: "$Date$"
    revision: "$Revision$"

class interface
    DATABASE [V1, V2, K]

create
    make

feature -- Commands

    add_record (v1: V1; v2: V2; k: K)
            -- Add a new record into current database.
        require
            non_existing_key: not exists (k)
        ensure
            record_added: values_1.has (v1) and values_2.has (v2) and exists (k)

    remove_record (k: K)
            -- Remove a record from current database.s
        require
            existing_key: exists (k)
        ensure
            database_count_decremented: count ~ (old count - 1)
```

```
                    key_removed: not exists (k)

feature -- Constructor

        make
                    -- Initialize an empty database.
                ensure
                    empty_database: keys.is_empty and values_1.is_empty and
values_2.is_empty
                    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

feature -- Queries

        count: INTEGER_32
                    -- Number of records in database.
                ensure
                    correct_result: Result = keys.count

        exists (k: K): BOOLEAN
                    -- Does key 'k' exist in the database?
                ensure
                    correct_result: Result = exists (k)

        get_keys (v1: V1; v2: V2): ITERABLE [K]
                    -- Keys that are associated with values 'v1' and 'v2'.
                ensure
                    result_contains_correct_keys_only: across
                            Result as x
                    all
                            (values_1.at (keys.index_of (x.item, 1)) ~ v1) and
(values_2.at (keys.index_of (x.item, 1)) ~ v2)
                    end
                    correct_keys_are_in_result: across
                            Current as db_cursor
                    all
                            if (db_cursor.item [2] ~ v1) and (db_cursor.item [3]
~ v2) and (across
                                Result as r_cursor
                            all
                                not exists (r_cursor.item)
                            end) then
                                False
                            else
                                True
                            end
                    end


feature -- alternative iteration cursor


        another_cursor: ITERATION_CURSOR [RECORD [V1, V2, K]]


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]]
```

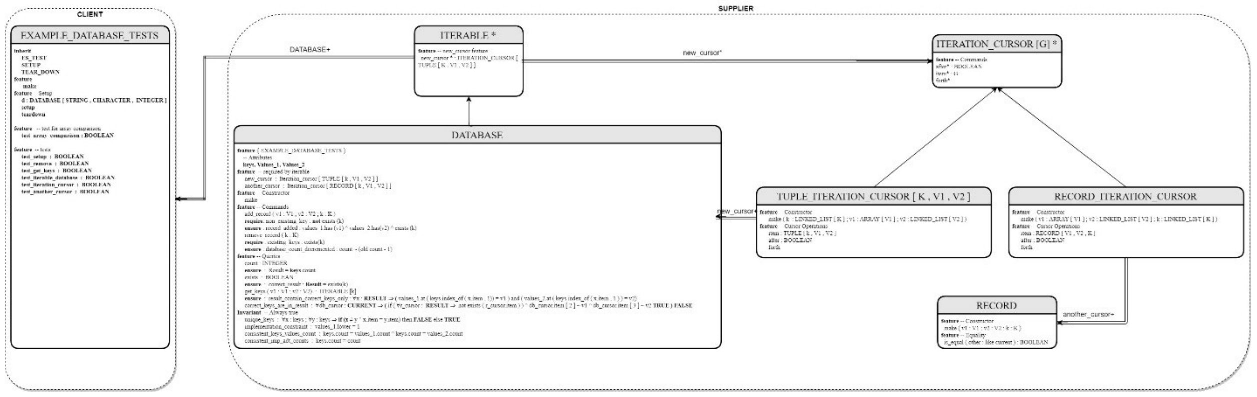
```
                    -- Fresh cursor associated with current structure

invariant
        unique_keys: across
                        keys as i
                all
                        across
                                keys as j
                        all
                                if i /~ j and i.item ~ j.item then
                                        False
                                else
                                        True
                                end
                        end
                end
        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 -- class DATABASE
note
        description: "Summary description for {RECORD}."
        author: "Fahad Qayyum"
        date: "$Date$"
        revision: "$Revision$"

class interface
        RECORD [V1, V2, K]
```

```
create
        make

feature -- Attributes (Do not modify this section)

        key: K

        value_1: V1

        value_2: V2

feature -- Commands (Do not modify this section)

        make (v1: V1; v2: V2; k: K)

feature -- Equality

        is_equal (other: like Current): BOOLEAN
                        -- Is `other` attached to an object considered
                        -- equal to current object?

end -- class RECORD
note
        description: "Summary description for {RECORD_ITERATION_CURSOR}."
        author: "Fahad Qayyum"
        date: "$Date$"
        revision: "$Revision$"

class
```

```
	RECORD_ITERATION_CURSOR [ V1, V2, K ]

inherit
	ITERATION_CURSOR [ RECORD [V1, V2, K] ]

create
	make

feature{NONE} --Atributes
	value_1_arr : ARRAY[V1]
	value_2_ll : LINKED_LIST[V2]
	key_ll : LINKED_LIST[K]
	cur_pos : INTEGER

feature
	make(v1 : ARRAY[V1]; v2 : LINKED_LIST[V2]; k : LINKED_LIST[K])
		do
			value_1_arr := v1
			value_2_ll := v2
			key_ll := k
			cur_pos := v1.lower
		end

feature -- Cursor Operations
	item : RECORD[V1, V2, K]
		do
			create result.make(value_1_arr[cur_pos], value_2_ll[cur_pos],
key_ll[cur_pos])
		end
	after : BOOLEAN
```

```
                do
                        Result := cur_pos > value_1_arr.upper
                end
        forth
                do
                        cur_pos := cur_pos + 1
                end
end
note
        description: "Summary description for {TUPLE_ITERATION_CURSOR}."
        author: "Fahad Qayyum"
        date: "$Date$"
        revision: "$Revision$"

class
        TUPLE_ITERATION_CURSOR [ K, V1, V2 ]

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

create
        make

feature{NONE} --Atributes
        value_1_arr : ARRAY[V1]
        value_2_ll : LINKED_LIST[V2]
        key_ll : LINKED_LIST[K]
        cur_pos : INTEGER

feature
```

```
        make( k : LINKED_LIST[K]; v1 : ARRAY[V1]; v2 : LINKED_LIST[V2])
                do
                        key_ll := k
                        value_1_arr := v1
                        value_2_ll := v2
                        cur_pos := value_1_arr.lower
                end

feature -- Cursor Operations
        item : TUPLE[K, V1, V2]
                local
                        value_1 : V1
                        value_2 : V2
                        key : K
                do
                        value_1 := value_1_arr[cur_pos]
                        value_2 := value_2_ll.at (cur_pos)
                        key := key_ll.at (cur_pos)
                        create result
                        Result :=  [ key , value_1 , value_2 ]
                end
        after : BOOLEAN
                do
                        Result := cur_pos > value_1_arr.upper
                end
        forth
                do
                        cur_pos := cur_pos + 1
                end
end
```



Q : explain how iterator pattern is implemented in the model cluster ?

Ans : The class DATABASE inherits ITERABLE[G] and hence iterable. Since the class ITERBALE[G] is a deferred class we must implement the feature in our sub class (DATABASE). The feature new_cursor is defined in class DATABASE and its return type is ITERATION_CURSOR [TUPLE [K , V1 , V2]]. In the implementation of new_cursor feature the dynamic type is set to TUPLE_ITERATION_CURSOR, which inherits ITERATION_CURSOR, a deferred class, and therefore, provides implementation for the features : ITEM, AFTER, FORTH . when iterating over the DATABASE class using new_cursor then the feature ITEM returns a TUPLE [K , V1 , V2]. And hence all the elements of the DATABASE are accessible through this iterator pattern.

Q : explain how you implemented the feature another cursor in the DATABASE model ?

Ans : The class DATABASE inherits ITERABLE[G] and hence iterable. Since the class ITERBALE[G] is a deferred class we must implement the feature in our sub class (DATABASE). The feature another_cursor is defined in class DATABASE and its return type is ITERATION_CURSOR [RECORD [V1 , V2 , K]]. In the implementation of another_cursor feature the dynamic type is set to RECORD_ITERATION_CURSOR, which inherits ITERATION_CURSOR, a deferred class, and therefore, provides implementation for the features : ITEM, AFTER, FORTH . when iterating over the DATABASE class using another_cursor then the feature ITEM returns a RECORD [K , V1 , V2]. And hence all the elements of the DATABASE are accessible through this iterator pattern.