EECS3311

Fall 2017

Saad Saeed

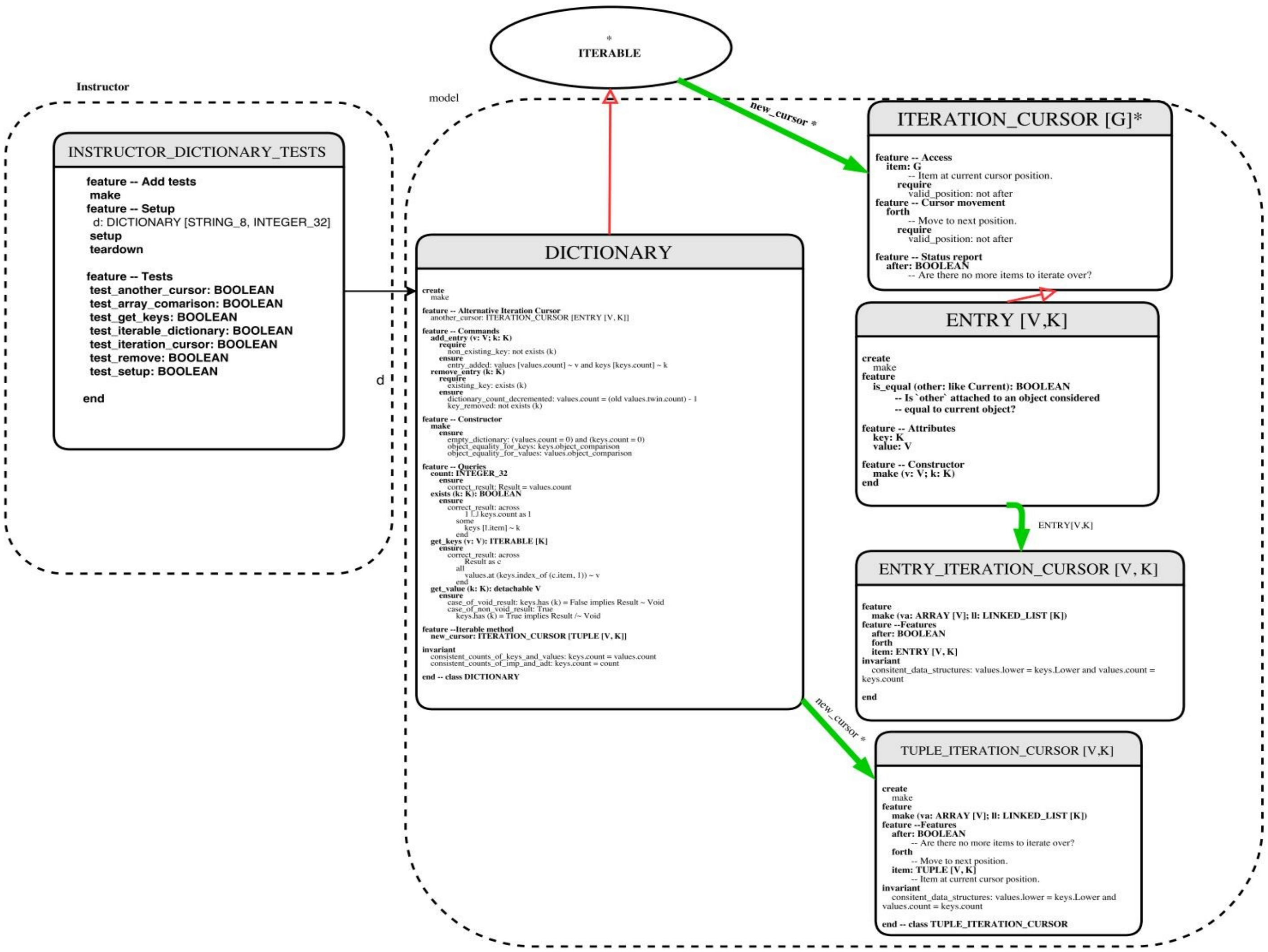
saeeds28

```
class interface
        DICTIONARY [V, K]

create
        make

feature -- Alternative Iteration Cursor

        another_cursor: ITERATION_CURSOR [ENTRY [V, K]]

feature -- Commands

        add_entry (v: V; k: K)
                        -- Add a new entry with key 'k' and value 'v'.
                        -- It is required that 'k' is not an existing search key in the dictionary.
                require
                        non_existing_key: not exists (k)
                ensure
                        entry_added: values [values.count] ~ v and keys [keys.count] ~ k

        remove_entry (k: K)
                        -- Remove the corresponding entry whose search key is 'k'.
                        -- It is required that 'k' is an existing search key in the dictionary.
                require
                        existing_key: exists (k)
                ensure
                        dictionary_count_decremented: values.count = (old values.twin.count) - 1
                        key_removed: not exists (k)
```

```
feature -- Constructor

          make
                    -- Initialize an empty dictionary.
               ensure
                    empty_dictionary: (values.count = 0) and (keys.count = 0)
                    object_equality_for_keys: keys.object_comparison
                    object_equality_for_values: values.object_comparison

feature -- Queries

          count: INTEGER_32
                    -- Number of entries in the dictionary.
               ensure
                    correct_result: Result = values.count

          exists (k: K): BOOLEAN
                    -- Does key 'k' exist in the dictionary?
               ensure
                    correct_result: across
                              1 |..| keys.count as l
                         some
                              keys [l.item] ~ k
                         end

          get_keys (v: V): ITERABLE [K]
                    -- Return an iterable collection of keys that are associated with value 'v'.
                    -- Hint: Refere to the architecture BON diagram of the Iterator Pattern, to see
                    -- what classes can be used to instantiate objects that are iterable.
```

```
        ensure
            correct_result: across
                    Result as c
                all
                    values.at (keys.index_of (c.item, 1)) ~ v
                end

    get_value (k: K): detachable V
            -- Return the assocated value of search key 'k' if it exists.
            -- Void if 'k' does not exist.
            -- Declaring "detachable" besides the return type here indicates that
            -- the return value might be void (i.e., null).
        ensure
            case_of_void_result: keys.has (k) = False implies Result ~ Void
            case_of_non_void_result: True
                keys.has (k) = True implies Result /~ Void

feature --Iterable method

    new_cursor: ITERATION_CURSOR [TUPLE [V, K]]
            -- Fresh cursor associated with current structure

invariant
    consistent_counts_of_keys_and_values: keys.count = values.count
    consistent_counts_of_imp_and_adt: keys.count = count

end --
```

Architectural Diagram



```
INSTRUCTOR_DICTIONARY_TESTS

  feature -- Add tests
   make
  feature -- Setup
   d: DICTIONARY [STRING_8, INTEGER_32]
   setup
   teardown

  feature -- Tests
   test_another_cursor: BOOLEAN
   test_array_comarison: BOOLEAN
   test_get_keys: BOOLEAN
   test_iterable_dictionary: BOOLEAN
   test_iteration_cursor: BOOLEAN
   test_remove: BOOLEAN
   test_setup: BOOLEAN

  end
```

```
DICTIONARY

create
   make

feature -- Alternative Iteration Cursor
   another_cursor: ITERATION_CURSOR [ENTRY [V, K]]

feature -- Commands
   add_entry (v: V; k: K)
      require
         non_existing_key: not exists (k)
      ensure
         entry_added: values [values.count] ~ v and keys [keys.count] ~ k
   remove_entry (k: K)
      require
         existing_key: exists (k)
      ensure
         dictionary_count_decremented: values.count = (old values.twin.count) - 1
         key_removed: not exists (k)

feature -- Constructor
   make
      ensure
         empty_dictionary: (values.count = 0) and (keys.count = 0)
         object_equality_for_keys: keys.object_comparison
         object_equality_for_values: values.object_comparison

feature -- Queries
   count: INTEGER_32
      ensure
         correct_result: Result = values.count
   exists (k: K): BOOLEAN
      ensure
         correct_result: across
               1 |..| keys.count as l
            some
               keys [l.item] ~ k
            end
   get_keys (v: V): ITERABLE [K]
      ensure
         correct_result: across
               Result as c
            all
               values.at (keys.index_of (c.item, 1)) ~ v
            end
   get_value (k: K): detachable V
      ensure
         case_of_void_result: keys.has (k) = False implies Result ~ Void
         case_of_non_void_result: True
            keys.has (k) = True implies Result /~ Void

feature --Iterable method
   new_cursor: ITERATION_CURSOR [TUPLE [V, K]]

invariant
   consistent_counts_of_keys_and_values: keys.count = values.count
   consistent_counts_of_imp_and_adt: keys.count = count

end -- class DICTIONARY
```

```
ITERATION_CURSOR [G]*

feature -- Access
   item: G
         -- Item at current cursor position.
      require
         valid_position: not after
feature -- Cursor movement
   forth
         -- Move to next position.
      require
         valid_position: not after

feature -- Status report
   after: BOOLEAN
         -- Are there no more items to iterate over?
```

```
ENTRY [V,K]

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

feature -- Attributes
   key: K
   value: V

feature -- Constructor
   make (v: V; k: K)
end
```

```
ENTRY_ITERATION_CURSOR [V, K]

feature
   make (va: ARRAY [V]; ll: LINKED_LIST [K])
feature --Features
   after: BOOLEAN
   forth
   item: ENTRY [V, K]
invariant
   consitent_data_structures: values.lower = keys.Lower and values.count =
keys.count

end
```

```
TUPLE_ITERATION_CURSOR [V,K]

create
   make
feature
   make (va: ARRAY [V]; ll: LINKED_LIST [K])
feature --Features
   after: BOOLEAN
         -- Are there no more items to iterate over?
   forth
         -- Move to next position.
   item: TUPLE [V, K]
         -- Item at current cursor position.
invariant
   consitent_data_structures: values.lower = keys.Lower and
values.count = keys.count

end -- class TUPLE_ITERATION_CURSOR
```

The iterator pattern is implemented as follows:
The Dictionary class is made to inherit from the ITERABLE class of type tuple[V,K]. The new_cursor method that is required in the class is implemented; it returns an ITERATION_CURSOR[TUPLE[V,K]] object. This object is an instantiation of the TUPLE_ITERATION_CURSOR class. This class inherits from ITERATION_CURSOR of type TUPLE[V,K]. Here the three required features item, forth, and after are defined. On the client side, the client calls the d.new_cursor method to get a new iterator.

The another_cursor method returns an ITERATION_CURSOR[ENTRY[V,K]] object. This object is an instantiation of the ENTRY_ITERATION_CURSOR class. This class inherits from ITERATION_CURSOR of type ENTRY[V,K]. Here the three required features item, forth, and after are defined. Furthermore, the method is_equal is redefined in the class ENTRY.