Lab 3: Report

EECS3311,
Fall 2017.
Michael Mierzwa.
213550702
mierzwam

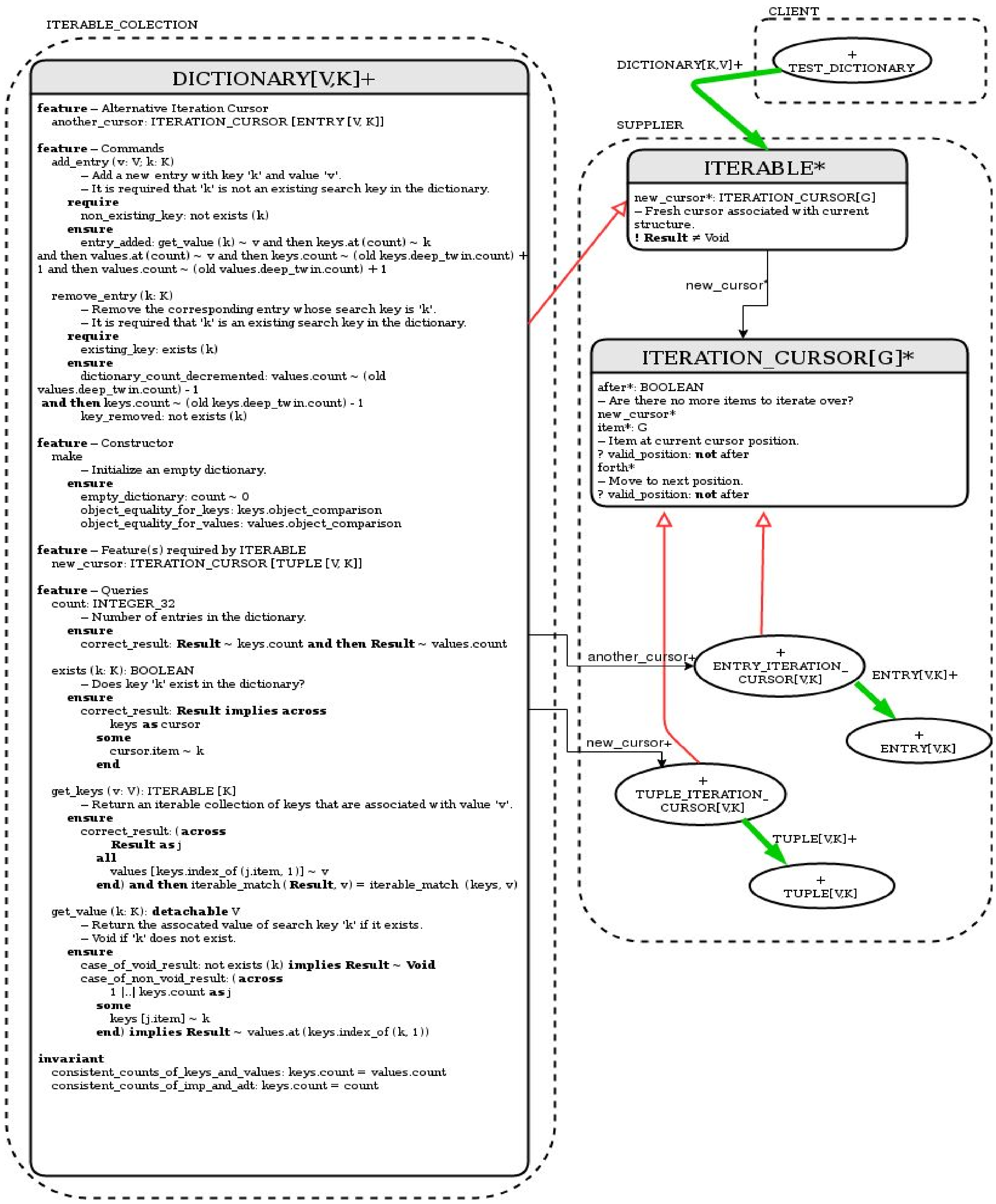
Section 1: Contract View

```
note
    description: "A Dictionary ADT mapping from keys to values"
    author: "Jackie and Michael Mierzwa"
    date: "$Date$"
    revision: "$Revision$"

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: get_value (k) ~ v and then
                         keys.at (count) ~ k and then
                         values.at (count) ~ v and then
                         keys.count ~ (old keys.deep_twin.count) + 1 and then
                         values.count ~ (old values.deep_twin.count) + 1

    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.deep_twin.count) - 1 and then
                 keys.count ~ (old keys.deep_twin.count) - 1
            key_removed: not exists (k)
```

```
feature -- Constructor

    make
             -- Initialize an empty dictionary.
       ensure
             empty_dictionary: count ~ 0
             object_equality_for_keys: keys.object_comparison
             object_equality_for_values: values.object_comparison

feature -- Feature(s) required by ITERABLE

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

feature -- Queries

    count: INTEGER_32
             -- Number of entries in the dictionary.
       ensure
             correct_result: Result ~ keys.count and then Result ~ values.count

    exists (k: K): BOOLEAN
             -- Does key 'k' exist in the dictionary?
       ensure
             correct_result: Result implies across
                             keys as cursor
                     some
                             cursor.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 j
                     all
                             values [keys.index_of (j.item, 1)] ~ v
                     end) and then
                    iterable_match (Result, v) = iterable_match (keys, v)

--note: iterable_match is a hidden helper feature that t
akes in an iterable and a value and returns an integer of how many times value
occurs in the iterable
```

```
    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: not exists (k) implies Result ~ Void
             case_of_non_void_result: (across
                         1 |..| keys.count as j
                   some
                         keys [j.item] ~ k
                   end) implies Result ~ values.at (keys.index_of (k, 1))

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

end -- class DICTIONARY
```

Section 2: Architectural Diagram



```
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: get_value (k) ~ v and then keys.at (count) ~ k
and then values.at (count) ~ v and then keys.count ~ (old keys.deep_twin.count) +
1 and then values.count ~ (old values.deep_twin.count) + 1

   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.deep_twin.count) - 1
 and then keys.count ~ (old keys.deep_twin.count) - 1
         key_removed: not exists (k)

feature – Constructor
   make
         – Initialize an empty dictionary.
      ensure
         empty_dictionary: count ~ 0
         object_equality_for_keys: keys.object_comparison
         object_equality_for_values: values.object_comparison

feature – Feature(s) required by ITERABLE
   new_cursor: ITERATION_CURSOR [TUPLE [V, K]]

feature – Queries
   count: INTEGER_32
         – Number of entries in the dictionary.
      ensure
         correct_result: Result ~ keys.count and then Result ~ values.count

   exists (k: K): BOOLEAN
         – Does key 'k' exist in the dictionary?
      ensure
         correct_result: Result implies across
               keys as cursor
            some
               cursor.item ~ k
            end

   get_keys (v: V): ITERABLE [K]
         – Return an iterable collection of keys that are associated with value 'v'.
      ensure
         correct_result: (across
               Result as j
            all
               values [keys.index_of (j.item, 1)] ~ v
            end) and then iterable_match (Result, v) = iterable_match  (keys, v)

   get_value (k: K): detachable V
         – Return the assocated value of search key 'k' if it exists.
         – Void if 'k' does not exist.
      ensure
         case_of_void_result: not exists (k) implies Result ~ Void
         case_of_non_void_result: (across
               1 |..| keys.count as j
            some
               keys [j.item] ~ k
            end) implies Result ~ values.at (keys.index_of (k, 1))

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

```
new_cursor*: ITERATION_CURSOR[G]
– Fresh cursor associated with current
structure.
! Result ≠ Void
```

```
after*: BOOLEAN
– Are there no more items to iterate over?
new_cursor*
item*: G
– Item at current cursor position.
? valid_position: not after
forth*
– Move to next position.
? valid_position: not after
```

The way I implemented the iterator pattern for model is to have DICTIONARY[K,V] be iterable with a default cursor (new_cursor) or an optional one (another_cursor). First in order for DICTIONARY to be iterable, I need to define a new ITERATION_CURSOR for it. The first cursor I define as new_cursor that returns an ITERATION_CURSOR, but I then say that this cursor is actually a child type of ITERATION_CURSOR known as TUPLE_ITERATION_CURSOR. This class defines the inherited after, item, and forth features as a representation of a pair of keys of type K and values of type V as a tuple. With this cursor implemented, we now give the client TEST_DICTIONARY the ability to iterate through the keys and values of DICTIONARY.

Another cursor is a slightly changed version of TUPLE_ITERATION_CURSOR, it uses ENTRY_ITERATION_CURSOR instead. The main difference between these two types of cursors is that it is of type ENTRY. How does that make it different? Well since we add some extra functionality for ENTRY by redefining is_equal, we can get a TUPLE that is more specialized for our needs, in this case we require two keys to be equal if they point to the same value. This definition of ENTRY allows us to better compare the key value pairs in dictionary.

END