Benjamin Korobkin

EECS 3311

Fall 2017

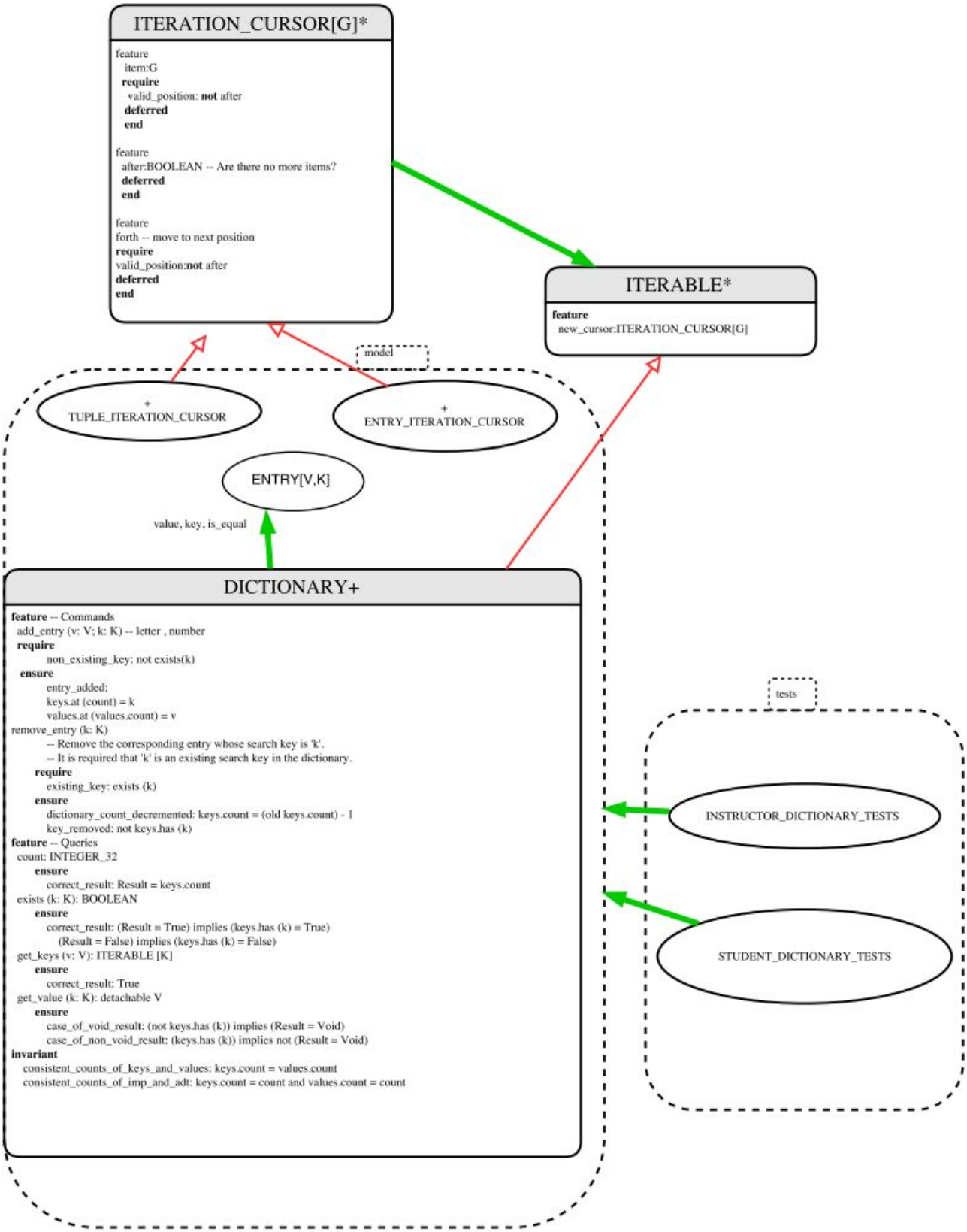
bjk125

Lab 3

Dictionary

```
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)
			-- letter , number
			-- 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: keys.at (count) = k
				values.at (values.count) = v

	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: keys.count = (old keys.count) - 1
			key_removed: not keys.has (k)

feature -- Constructor
	make
		ensure
			empty_dictionary: values.is_empty and keys.is_empty
			object_equality_for_keys: keys.object_comparison
			object_equality_for_values: values.object_comparison

feature -- Feature(s) required by ITERABLE
-- Your Task
-- See test_iterable_dictionary and test_iteration_cursor in
INSTRUCTOR_DICTIONARY_TESTS.
-- As soon as you make the current class iterable,
```

```
-- define the necessary feature(s) here.

	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

	exists (k: K): BOOLEAN
			-- Does key 'k' exist in the dictionary?
		ensure
			correct_result: (Result = True) implies (keys.has (k) = True)
				(Result = False) implies (keys.has (k) = False)

	get_keys (v: V): ITERABLE [K]
			-- Return an iterable collection of keys that are associated with value 'v'.
			-- Hint: Refer to the architecture BON diagram of the Iterator Pattern, to
see
			-- what classes can be used to instantiate objects that are iterable. -- we
have keys as a linkedlist given to us already
		ensure
			correct_result: True

	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 keys.has (k)) implies (Result = Void)
			case_of_non_void_result: (keys.has (k)) implies not (Result = Void)

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

end -- class DICTIONARY
```


ITERATION_CURSOR[G]*
feature
item:G
require
valid_position: not after
deferred
end
feature
after:BOOLEAN -- Are there no more items?
deferred
end
feature
forth -- move to next position
require
valid_position:not after
deferred
end
ITERABLE*
feature
new_cursor:ITERATION_CURSOR[G]
model
+
TUPLE_ITERATION_CURSOR
+
ENTRY_ITERATION_CURSOR
ENTRY[V,K]
value, key, is_equal
DICTIONARY+
feature -- Commands
add_entry (v: V; k: K) -- letter , number
require
non_existing_key: not exists(k)
ensure
entry_added:
keys.at (count) = k
values.at (values.count) = v
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: keys.count = (old keys.count) - 1
key_removed: not keys.has (k)
feature -- Queries
count: INTEGER_32
ensure
correct_result: Result = keys.count
exists (k: K): BOOLEAN
ensure
correct_result: (Result = True) implies (keys.has (k) = True)
(Result = False) implies (keys.has (k) = False)
get_keys (v: V): ITERABLE [K]
ensure
correct_result: True
get_value (k: K): detachable V
ensure
case_of_void_result: (not keys.has (k)) implies (Result = Void)
case_of_non_void_result: (keys.has (k)) implies not (Result = Void)
invariant
consistent_counts_of_keys_and_values: keys.count = values.count
consistent_counts_of_imp_and_adt: keys.count = count and values.count = count
tests
INSTRUCTOR_DICTIONARY_TESTS
STUDENT_DICTIONARY_TESTS

Korobkin 212231189

The iterator pattern is implemented in the model cluster by having dictionary entries kept in tuples of values and keys, corresponding to words and numbers. The entry are derived from class ENTRY where the values, keys, and the is_equal method are initiated. The DICTIONARY class uses both entry and tuple iteration cursors to run through its entries. This is done in the iteration_cursor classes by first inheriting from the ITERATION_CURSOR class and then redefining the iteration features (item, forth, and after) in their respective classes. The DICTIONARY class calls these to initialize its own cursors for itself.
another_cursor is implemented by first creating a local cursor with generic parameters (V and K) and then it is initialized with (values, keys) that are created in the ENTRY_ITERATION_CURSOR class. This method allows us to use our own entry cursor that is specific to our needs rather than relying on a more generic tuple iteration cursor.