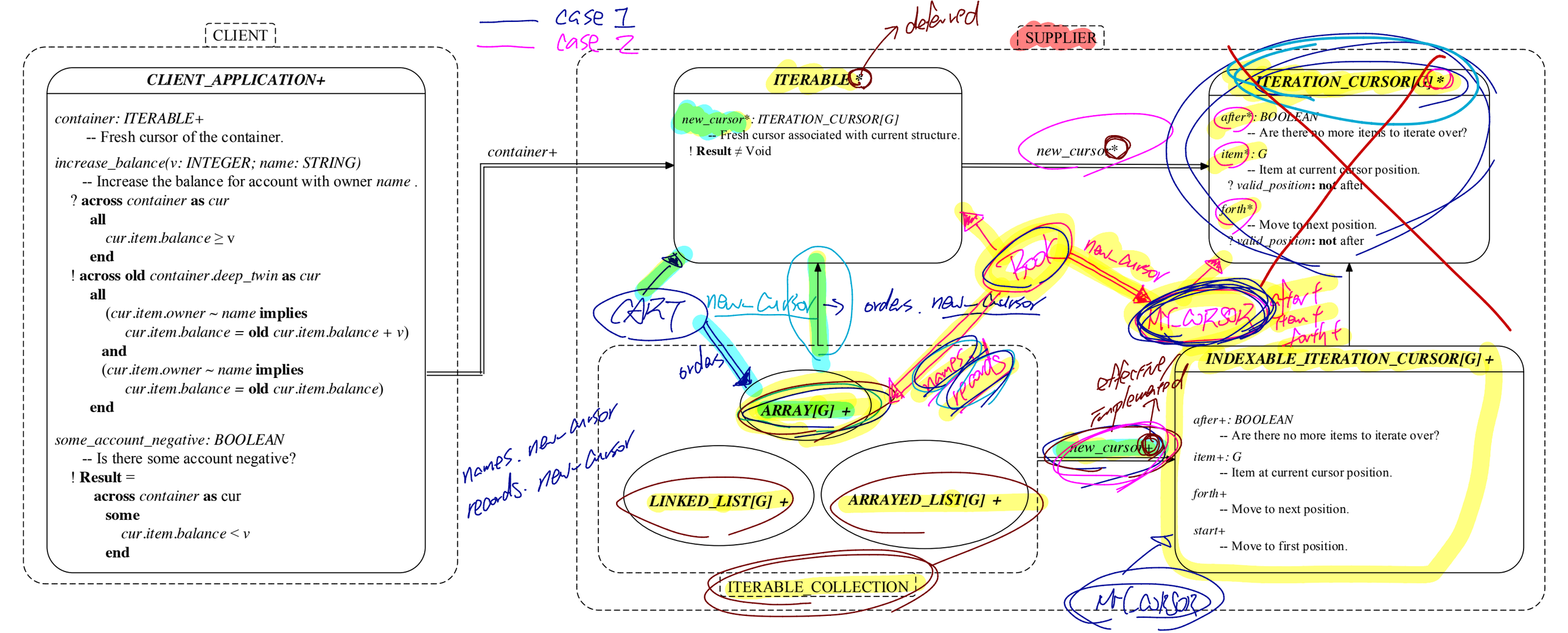
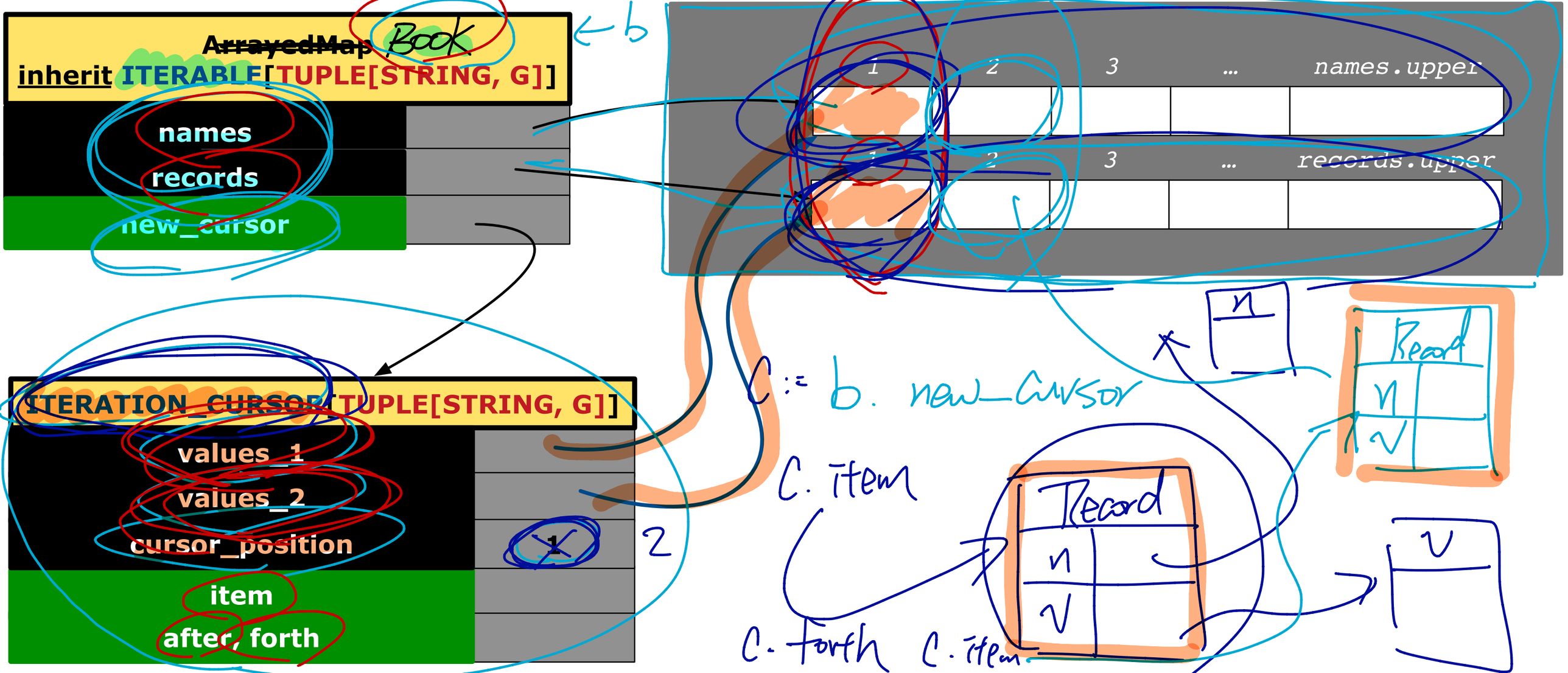
Thursday Sep. 27

Lecture 7


CLIENT
CLIENT_APPLICATION+
container: ITERABLE+
-- Fresh cursor of the container.
increase_balance(v: INTEGER; name: STRING)
-- Increase the balance for account with owner name .
? across container as cur
all
cur.item.balance ≥ v
end
! across old container.deep_twin as cur
all
(cur.item.owner ~ name implies
cur.item.balance = old cur.item.balance + v)
and
(cur.item.owner ~ name implies
cur.item.balance = old cur.item.balance)
end
some_account_negative: BOOLEAN
-- Is there some account negative?
! Result =
across container as cur
some
cur.item.balance < v
end
case 1
case 2
deferred
SUPPLIER
ITERABLE*
new_cursor*: ITERATION_CURSOR[G]
-- Fresh cursor associated with current structure.
! Result ≠ Void
container+
new_cursor*
ITERATION_CURSOR[G]*
after*: BOOLEAN
-- Are there no more items to iterate over?
item*: G
-- Item at current cursor position.
? valid_position: not after
forth*
-- Move to next position.
? valid_position: not after
CART
new_cursor
orders.new_cursor
Rock
new_cursor
MY_CURSOR
after f
item f
forth f
orders
ARRAY[G] +
names
records
names.new_cursor
records.new_cursor
LINKED_LIST[G] +
ARRAYED_LIST[G] +
ITERABLE_COLLECTION
effective
implemented
new_cursor+
INDEXABLE_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.
start+
-- Move to first position.
MY_CURSOR

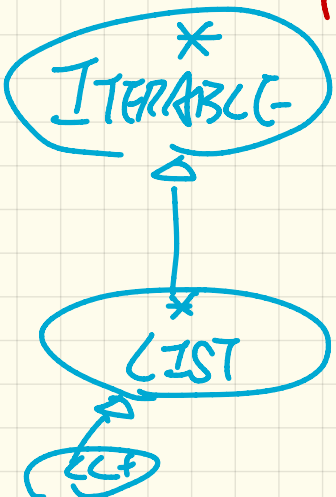

ArrayedMap
Book
inherit ITERABLE[TUPLE[STRING, G]]
names
records
new_cursor
b
1
2
3
...
names.upper
records.upper
ITERATION_CURSOR[TUPLE[STRING, G]]
values_1
values_2
cursor_position
item
after, forth
2
c := b.new_cursor
c.item
c.forth
c.item
n
v
Record

```
class
  CHECKER
feature -- Attributes
  collection: ITERABLE [INTEGER]
feature -- Queries
  is_all_positive: BOOLEAN
      -- Are all items in collection positive?
    do
      ...
    ensure
      across
        collection as cursor
      all
        cursor.item > 0
      end
  end
```

Inv

Programming with Interface vs. Implementation

change LIST to ITERABLE?




```
class BANK
...
  accounts: ITERABLE [ACCOUNT]
  binary_search (acc_id: INTEGER): ACCOUNT
      -- Search on accounts sorted in non-descending order.
    require
      across
        1 |..| (accounts.count - 1) as cursor
      all
        accounts [cursor.item].id <= accounts [cursor.item + 1].id
      end
    do
      ...
    ensure
      Result.id = acc_id
    end
```

Across

Use of ITERABLE vs. ITERATION_CURSOR

```
class BANK
 accounts: ITERABLE [ACCOUNT]
 max_balance: ACCOUNT
    -- Account with the maximum balance value.
  require ??
  local
   cursor: ITERATION_CURSOR[ACCOUNT]; max: ACCOUNT
  do
   from max := accounts [1]; cursor := accounts.new_cursor
   until cursor.after
   do
    if cursor.item.balance > max.balance then
      max := cursor.item
    end
    cursor.forth
   end
  ensure ??
  end
```

across accounts as cursor
loop
  f . . .
  cursor.item
  X cursor.forth
end

What's the equivalent use of across without any mention of the iteration cursor?

Non-Once Query

"Alan" "Mark"

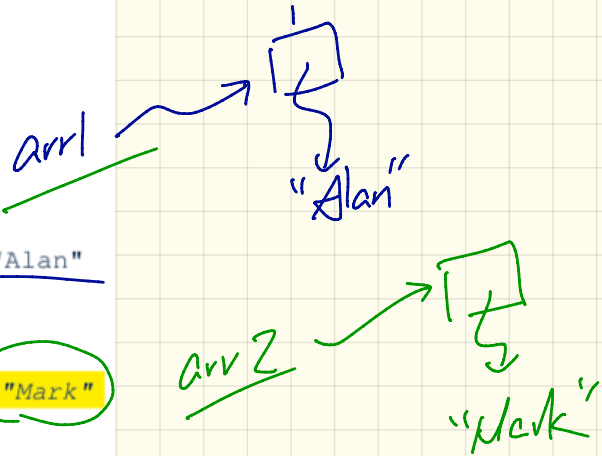
```
new_array (s: STRING): ARRAY[STRING]
    -- An ordinary query that returns an array.
  do
    create {ARRAY[STRING]} Result.make_empty
    Result.force (s, Result.count + 1)
  end
```

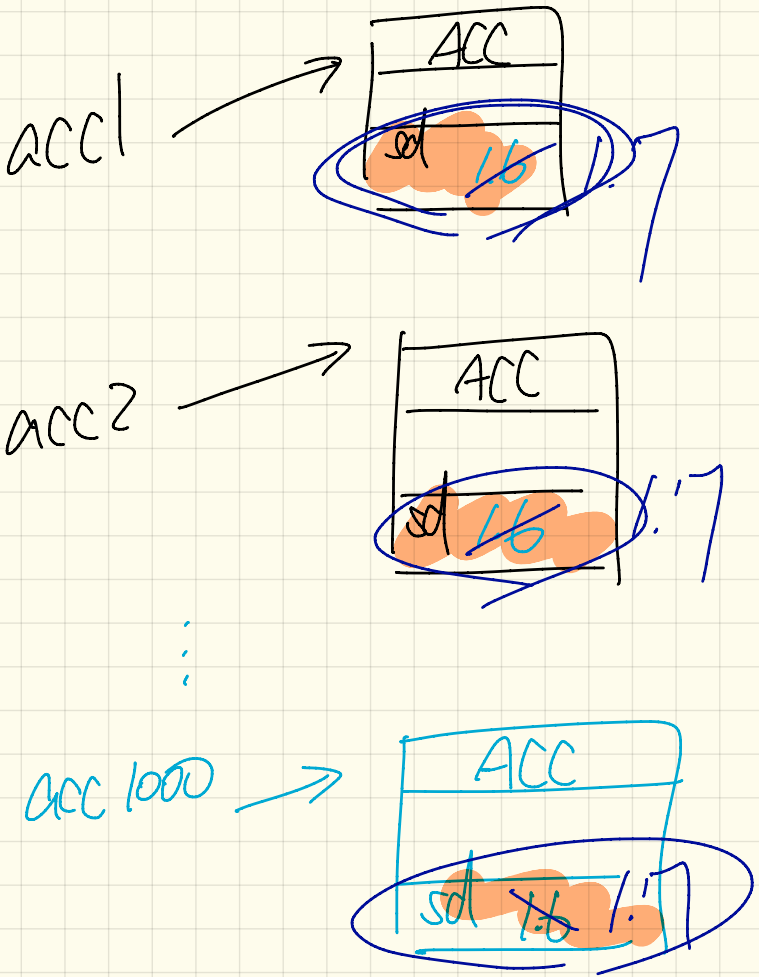
```
test_query: BOOLEAN
  local
    a: A
    arr1, arr2: ARRAY[STRING]
  do
    create a.make

    arr1 := a.new_array ("Alan")
    Result := arr1.count = 1 and arr1[1] ~ "Alan"
    check Result end

    arr2 := a.new_array ("Mark")
    Result := arr2.count = 1 and arr2[1] ~ "Mark"
    check Result end

    Result := not (arr1 = arr2)
    check Result end
  end
```





1.6 ↑ 1.7

Once - Query

```
new_once_array (s: STRING): ARRAY[STRING]
    -- A once query that returns an array.
  once
    create {ARRAY[STRING]} Result.make_empty
    Result.force (s, Result.count + 1)
  end
```

```
test_once_query: BOOLEAN
  local
    a: A
    arr1, arr2: ARRAY[STRING]
  do
    create a.make
    arr1 := a.new_once_array ("Alan")
    Result := arr1.count = 1 and arr1[1] ~ "Alan"
    check Result end
    arr2 := a.new_once_array ("Mark")
    Result := arr2.count = 1 and arr2[1] ~ "Alan"
    check Result end

    Result := arr1 = arr2
    check Result end
end
```