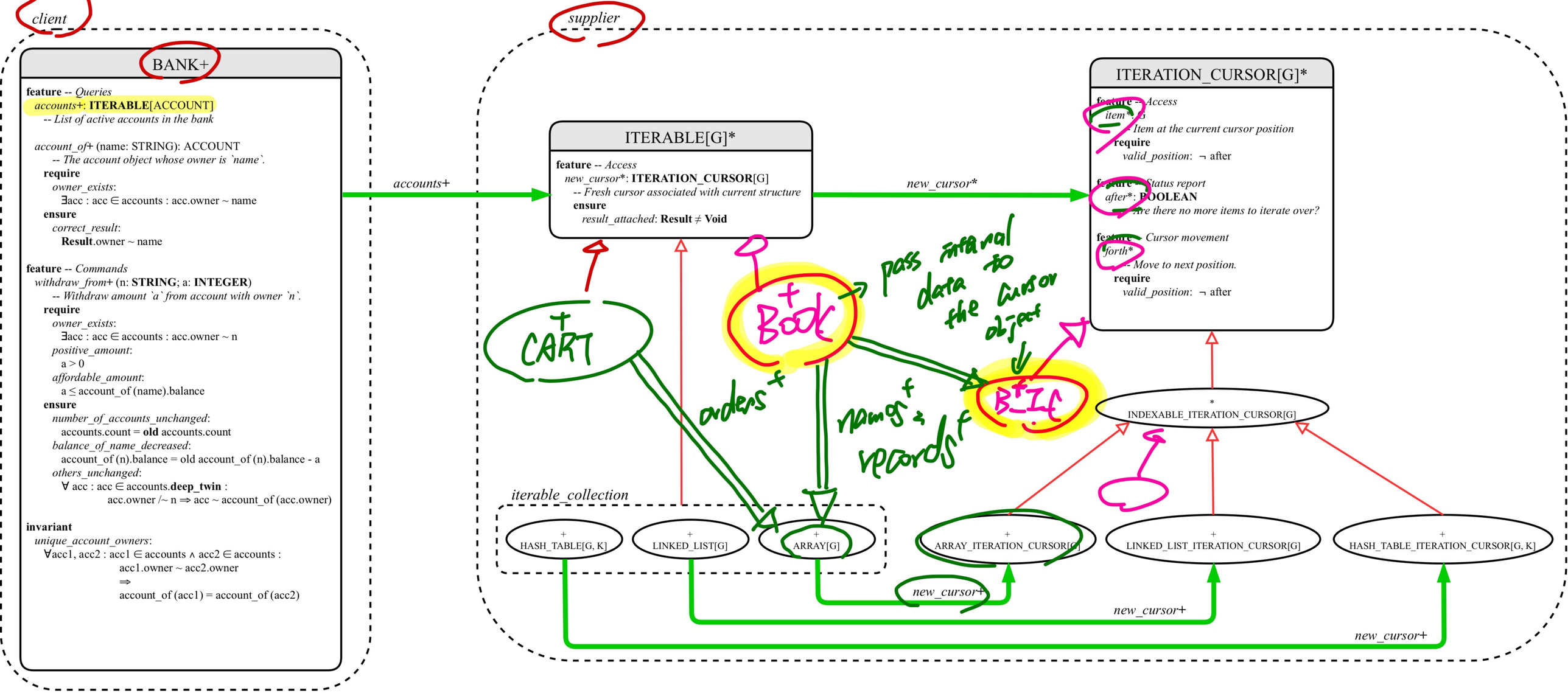
Lecture 7

Thursday September 26


client
supplier
BANK+
feature -- Queries
accounts+: ITERABLE[ACCOUNT]
-- List of active accounts in the bank
account_of+ (name: STRING): ACCOUNT
-- The account object whose owner is `name`.
require
owner_exists:
∃acc : acc ∈ accounts : acc.owner ~ name
ensure
correct_result:
Result.owner ~ name
feature -- Commands
withdraw_from+ (n: STRING; a: INTEGER)
-- Withdraw amount `a` from account with owner `n`.
require
owner_exists:
∃acc : acc ∈ accounts : acc.owner ~ n
positive_amount:
a > 0
affordable_amount:
a ≤ account_of (name).balance
ensure
number_of_accounts_unchanged:
accounts.count = old accounts.count
balance_of_name_decreased:
account_of (n).balance = old account_of (n).balance - a
others_unchanged:
∀ acc : acc ∈ accounts.deep_twin :
acc.owner /~ n ⇒ acc ~ account_of (acc.owner)
invariant
unique_account_owners:
∀acc1, acc2 : acc1 ∈ accounts ∧ acc2 ∈ accounts :
acc1.owner ~ acc2.owner
⇒
account_of (acc1) = account_of (acc2)
accounts+
ITERABLE[G]*
feature -- Access
new_cursor*: ITERATION_CURSOR[G]
-- Fresh cursor associated with current structure
ensure
result_attached: Result ≠ Void
new_cursor*
ITERATION_CURSOR[G]*
feature -- Access
item*: G
-- Item at the current cursor position
require
valid_position: ¬ after
feature -- Status report
after*: BOOLEAN
-- Are there no more items to iterate over?
feature -- Cursor movement
forth*
-- Move to next position.
require
valid_position: ¬ after
CART
Book
B_It
pass interval to data the cursor object
orders+
names+ & records+
iterable_collection
HASH_TABLE[G, K]
LINKED_LIST[G]
ARRAY[G]
INDEXABLE_ITERATION_CURSOR[G]
ARRAY_ITERATION_CURSOR[G]
LINKED_LIST_ITERATION_CURSOR[G]
HASH_TABLE_ITERATION_CURSOR[G, K]
new_cursor+
new_cursor+
new_cursor+

Implementing the Iterator Pattern: Easy Case

```
class
  CART
inherit
  ITERABLE[ORDER]

...

feature {NONE} -- Information Hiding
  orders: ARRAY[ORDER]

feature -- Iteration
  new_cursor: ITERATION_CURSOR[ORDER]
    do
      Result := orders.new_cursor
    end
```

Implementing the Iterator Pattern: Hard Case (1)

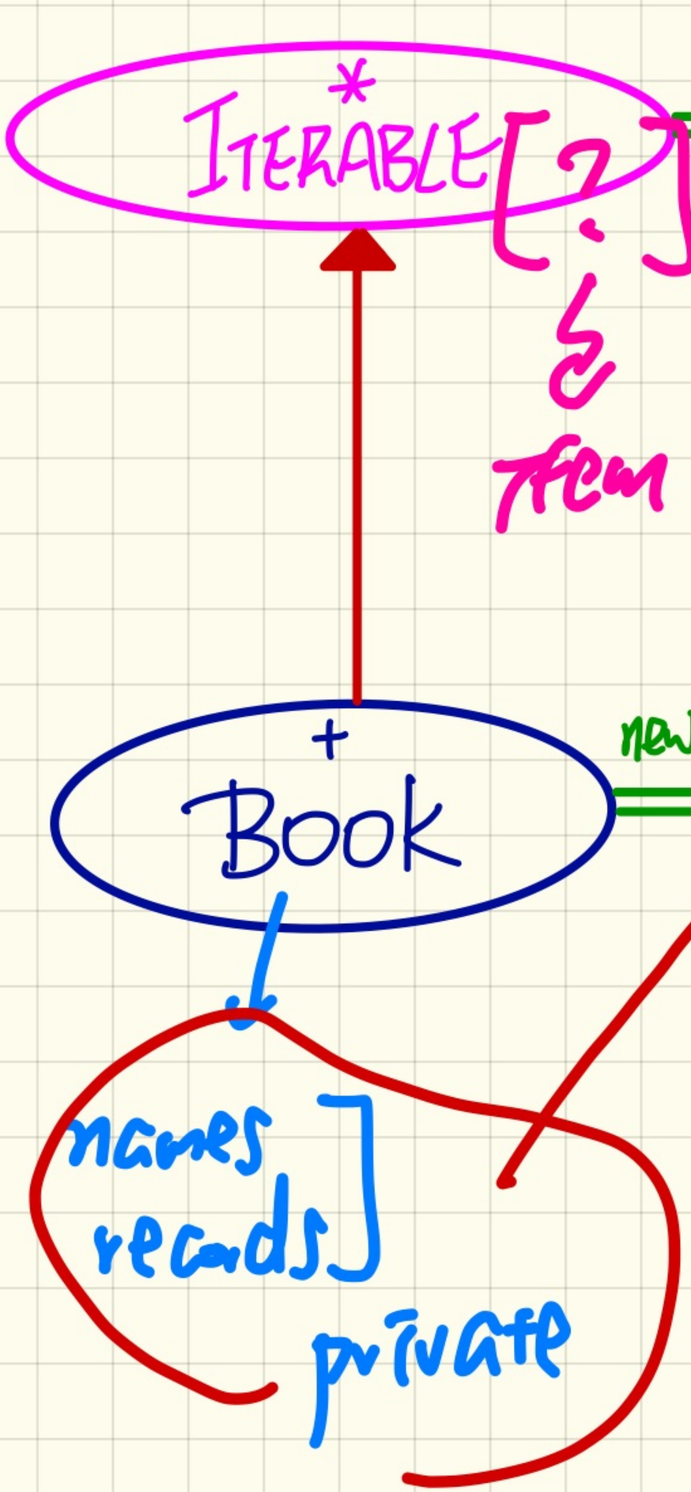
```
class
  GENERIC_BOOK[G]
inherit
  ITERABLE[ TUPLE[STRING, G] ]
...
feature {NONE} -- Information Hiding
  names: ARRAY[STRING]
  records: ARRAY[G]
feature -- Iteration
  new_cursor: ITERATION_CURSOR[ TUPLE[STRING, G] ]
    local
      cursor: MY_ITERATION_CURSOR[G]
    do
      create cursor.make (names, records)
      Result := cursor
    end
```

names. n_C

nRecords. n_C

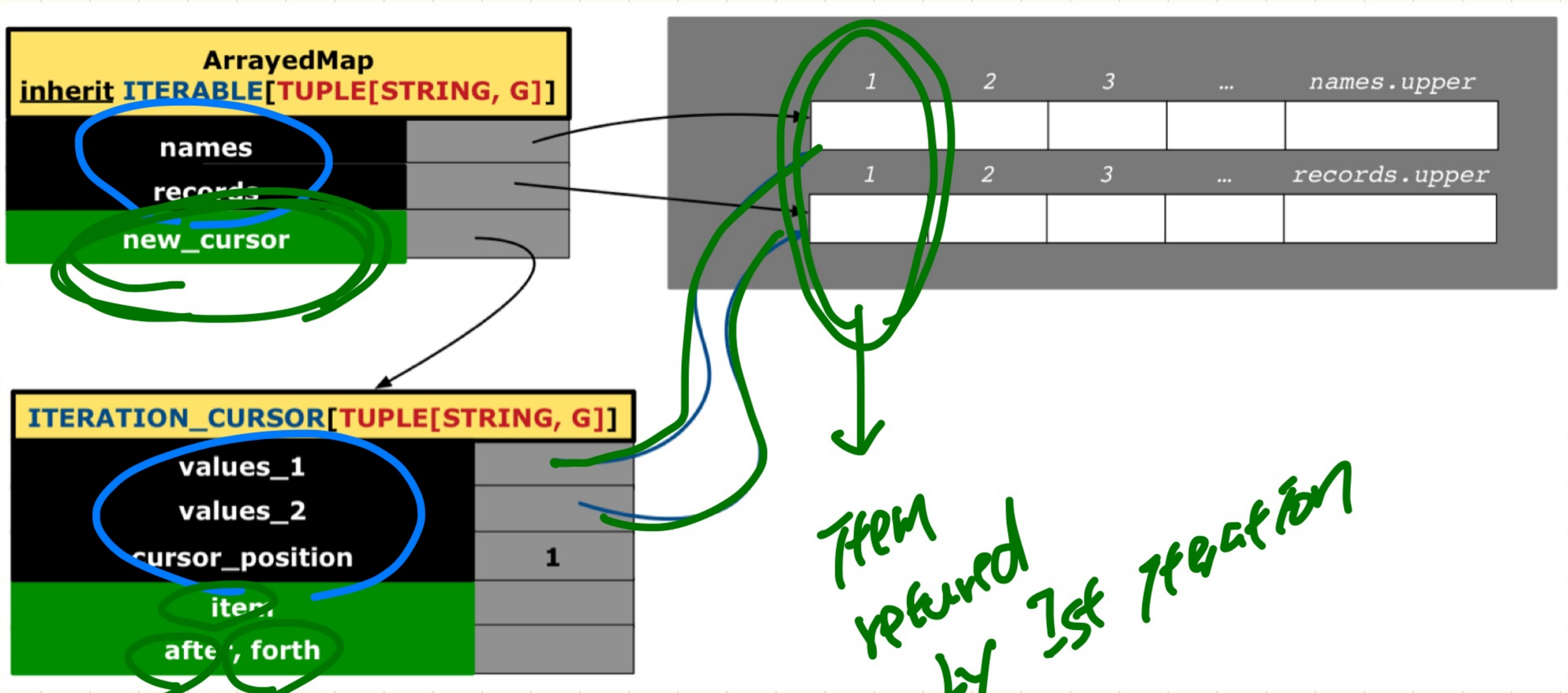
Item returned by each iteration of the cursor

Implementing the Iterator Pattern: Hard Case (2)




```
class
  MY_ITERATION_CURSOR[G]
inherit
  ITERATION_CURSOR[ TUPLE[STRING, G] ]
feature -- Constructor
  make (ns: ARRAY[STRING]; rs: ARRAY[G])
    do ... end
feature {NONE} -- Information Hiding
  cursor_position: INTEGER
  names: ARRAY[STRING]
  records: ARRAY[G]
feature -- Cursor Operations
  item: TUPLE[STRING, G]
    do ... end
  after: Boolean
    do ... end
  forth
    do ... end
```

Iterator Pattern at Runtime



Use of Iterable in Contracts

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

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

across |1..| 5| is i

Io.put_int(i)

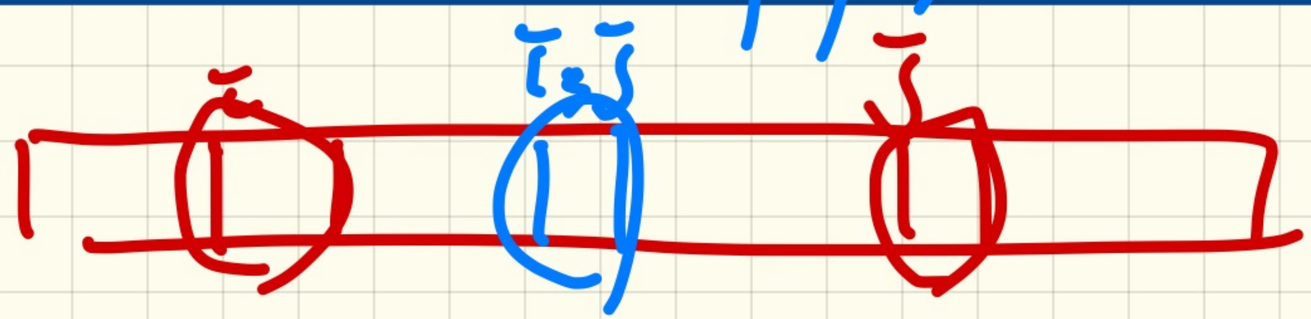
6-i

end

1	5
2	4
3	3
4	2
5	1

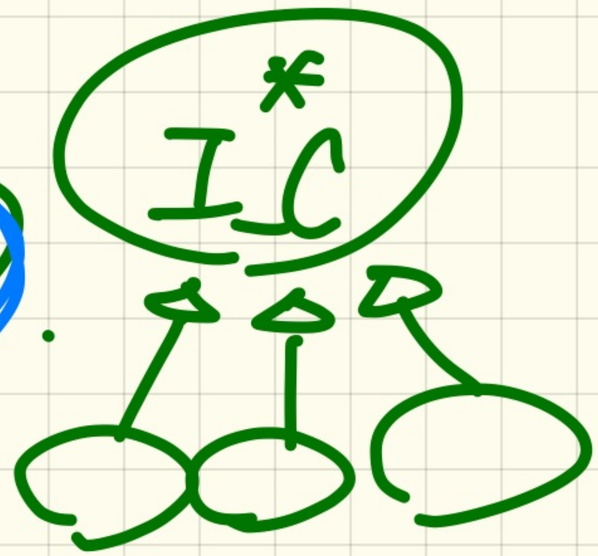
Use of Iterable in Contracts: Exercise

```
class BANK
...
  accounts: LIST [ACCOUNT]
  contains_duplicate: BOOLEAN
      -- Does the account list contain duplicate?
    do
      ...
ensure
    ∀ i, j ∈ INTEGER |
      1 ≤ i ≤ accounts.count ∧ 1 ≤ j ≤ accounts.count •
        accounts[i] ~ accounts[j] ⇒ i = j
end
```



Use of Iterable in Implementation (1)

```
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
```

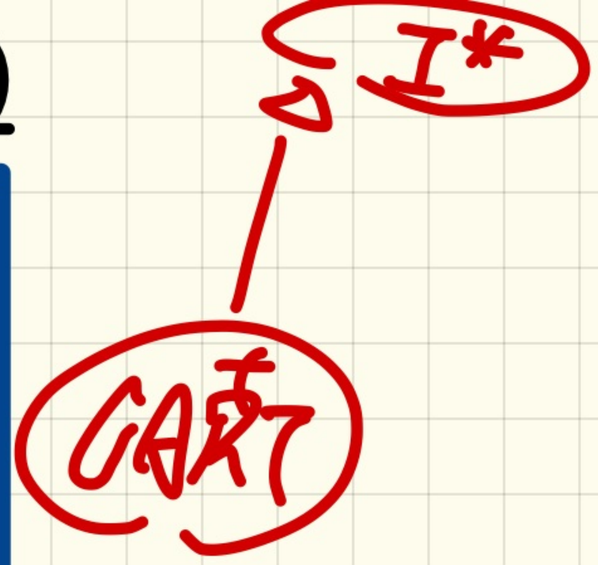


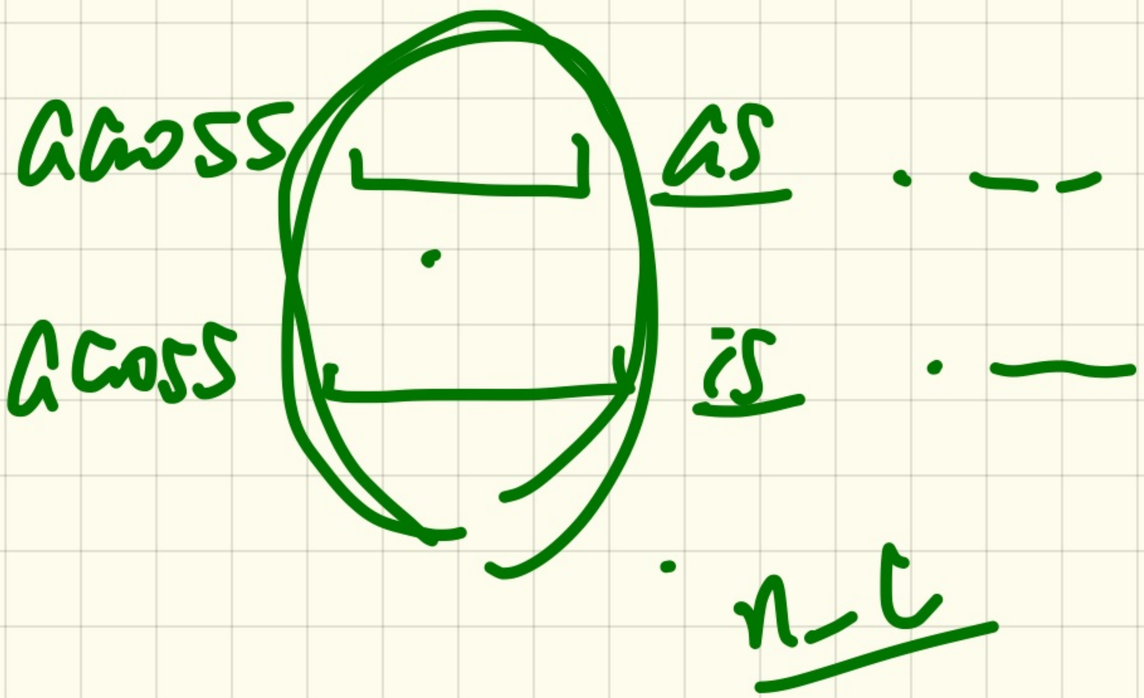
I_C. → return a new iteration cursor for 'accounts' in the starting pos.

move cursor to next item

Use of Iterable in Implementation (2)

```
class SHOP
  cart: CART
  checkout: INTEGER
      -- Total price calculated based on orders in the cart.
    require
      ??
    do
      across
        cart is order
      loop
        Result := Result + order.price * order.quantity
      end
    ensure
      ??
  end
```






```
class BANK
  accounts: ITERABLE [ACCOUNT]
  max_balance: ACCOUNT
      -- Account with the maximum balance value.
    require
      ??
    local
      max: ACCOUNT
    do
      max := accounts [1]
      across
        accounts is acc
      loop
        if acc.balance > max.balance then
          max := acc
        end
      end
    ensure
      ??
  end
```

Exercise 1

```
deferred class
  ITERABLE [G]
feature -- Access
  new_cursor: ITERATION_CURSOR [G]
    deferred end
end
```

new_cursor*

```
deferred class
  ITERATION_CURSOR [G]
feature -- Cursor features
  item: G
    deferred end

  after: BOOLEAN
    deferred end

  forth
    deferred end
```

```
test_database: BOOLEAN
  local
    db: DATABASE[STRING, INTEGER]
    tuples: LINKED_LIST[TUPLE[INTEGER, STRING]]
  do
    create db.make
    create tuples.make
    across
      db is t
    loop
      tuples.extend (t)
    end
end
```

```
class
  DATABASE[G, H]
inherit
  ITERABLE[TUPLE[H, G]]
feature {NONE} -- Implementation
  gs: ARRAY[G]
  hs: ARRAY[H]
feature -- Iterable
  new_cursor: ITERATION_CURSOR[TUPLE[H, G]]
    local
      db_cursor: ITEM_ITERATION_CURSOR[H, G]
    do
      create db_cursor.make (gs, hs)
      Result := db_cursor
    end
end
```

new_cursor+

```
class
  ITEM_ITERATION_CURSOR[M, N]
inherit
  ITERATION_CURSOR[TUPLE[M, N]]
create
  make
feature {NONE} -- Implementation
  ms: ARRAY[M]
  ns: ARRAY[N]
feature -- Constructor
  make (new_ns: ARRAY[N]; new_ms: ARRAY[M])
    do ... end
feature -- Cursor features
  item: TUPLE[M, N]
    do ... end

  after: BOOLEAN
    do ... end

  forth
    do ... end
end
```

```
deferred class
  ITERABLE [G]
feature -- Access
  new_cursor: ITERATION_CURSOR [G]
    deferred end
end
```

new_cursor*

```
deferred class
  ITERATION_CURSOR [G]
feature -- Cursor features
  item: G
    deferred end

  after: BOOLEAN
    deferred end

  forth
    deferred end
```

```
test_database: BOOLEAN
  local
    db: DATABASE[STRING, INTEGER]
    tuples: LINKED_LIST[TUPLE[INTEGER, STRING]]
  do
    create db.make
    create tuples.make
    across
      db is t
    loop
      tuples.extend (t)
    end
end
```

Exercise 2

Still compiles?

```
class
  DATABASE[G, H]
inherit
  ITERABLE[TUPLE[H, G]]
feature {NONE} -- Implementation
  gs: ARRAY[G]
  hs: ARRAY[H]
feature -- Iterable
  new_cursor: ITERATION_CURSOR[TUPLE[H, G]]
    local
      db_cursor: ITEM_ITERATION_CURSOR[H, G]
    do
      create db_cursor.make (gs, hs)
      Result := db_cursor
    end
end
```

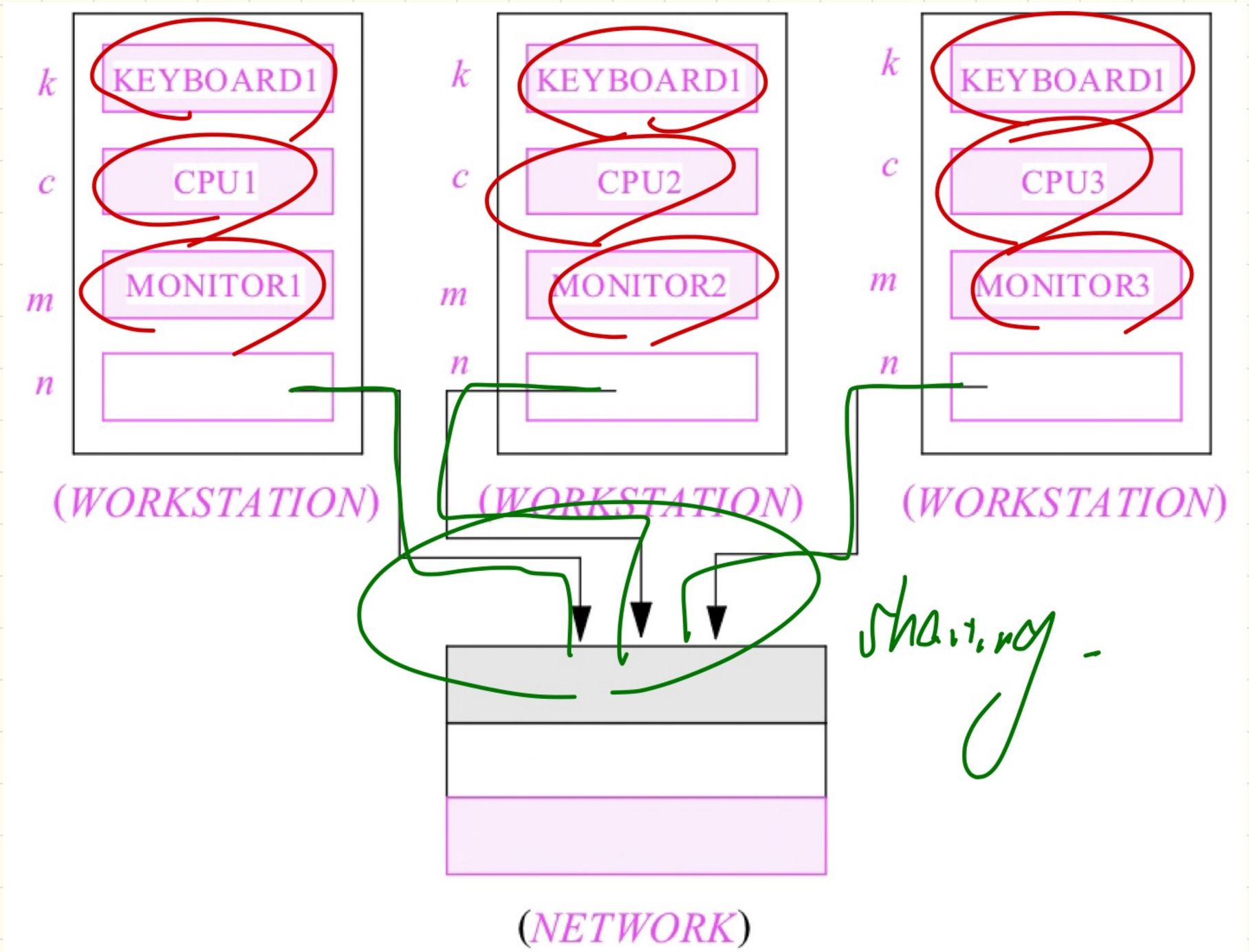
new_cursor+

```
class
  ITEM_ITERATION_CURSOR[M, N]
inherit
  ITERATION_CURSOR[TUPLE[M, N]]
create
  make
feature {NONE} -- Implementation
  ms: ARRAY[M]
  ns: ARRAY[N]
feature -- Constructor
  make (new_ns: ARRAY[N]; new_ms: ARRAY[M])
    do ... end
feature -- Cursor features
  item: TUPLE[M, N]
    do ... end

  after: BOOLEAN
    do ... end

  forth
    do ... end
end
```

Modelling: Aggregation vs. Composition



Use of Expanded Type

```
expanded class
  B
feature
  change_i (ni: INTEGER)
    do
      i := ni
    end
feature
  i: INTEGER
end
```

```
1   test_expanded: BOOLEAN
2     local
3       eb1, eb2: B
4     do
5       Result := eb1.i = 0 and eb2.i = 0
6       check Result end
7       Result := eb1 = eb2
8       check Result end
9       eb2.change_i (15)
10      Result := eb1.i = 0 and eb2.i = 15
11      check Result end
12      Result := eb1 /= eb2
13      check Result end
14    end
```

for expanded type, compare contents.

Q: ARRAY[INT]