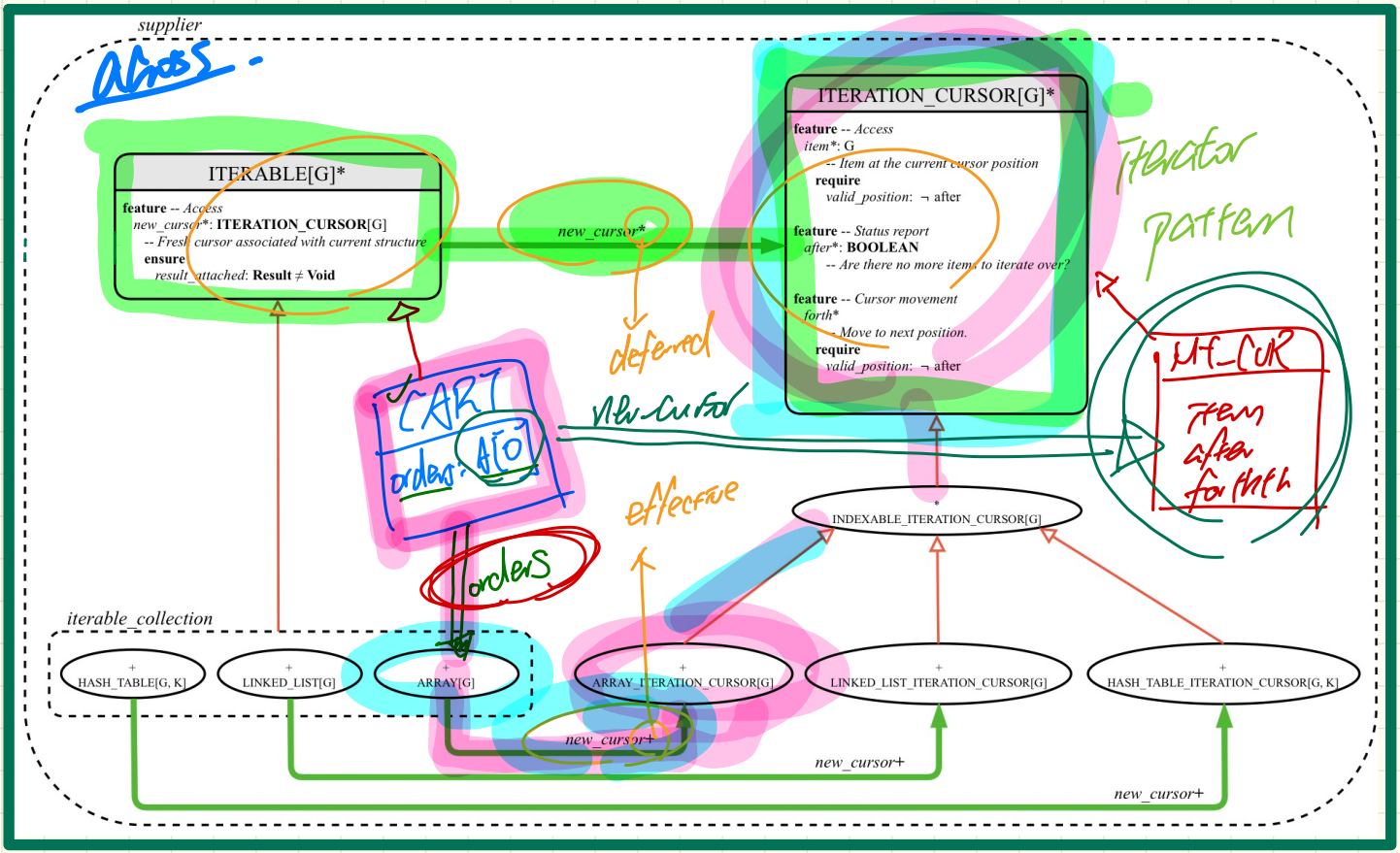
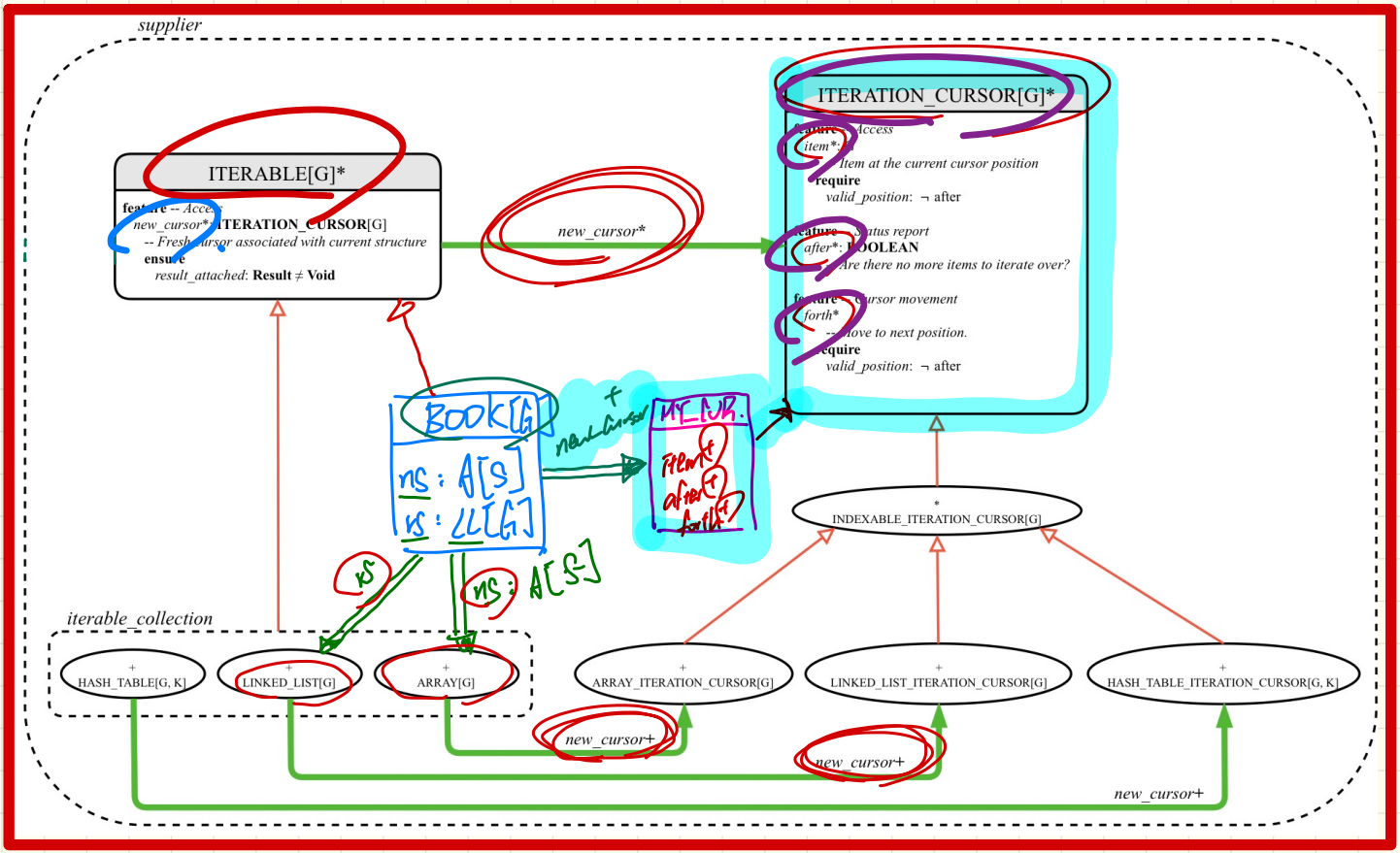
Lecture 11

Monday February 10

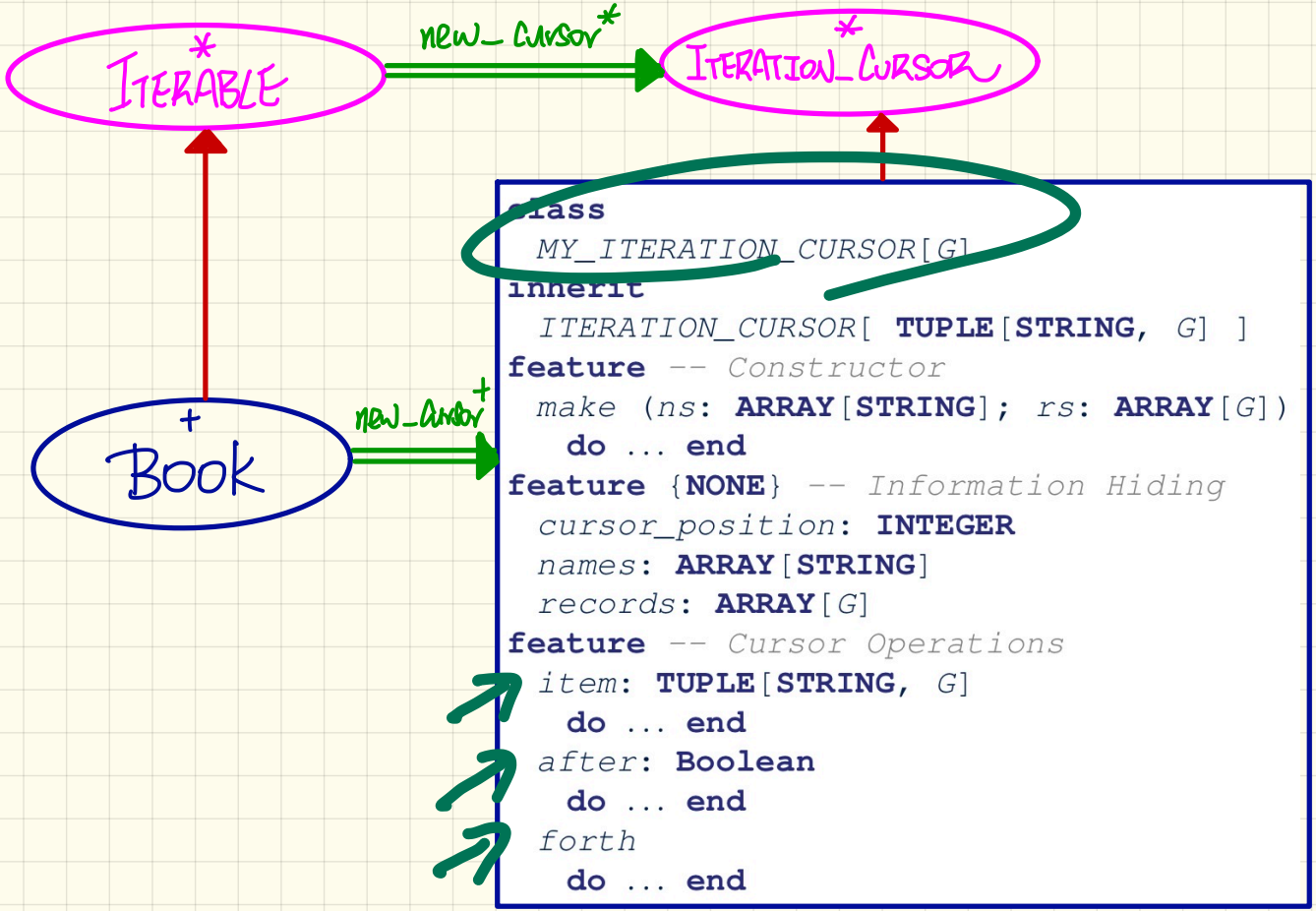
Implementing the Iterator Pattern: Easy Case



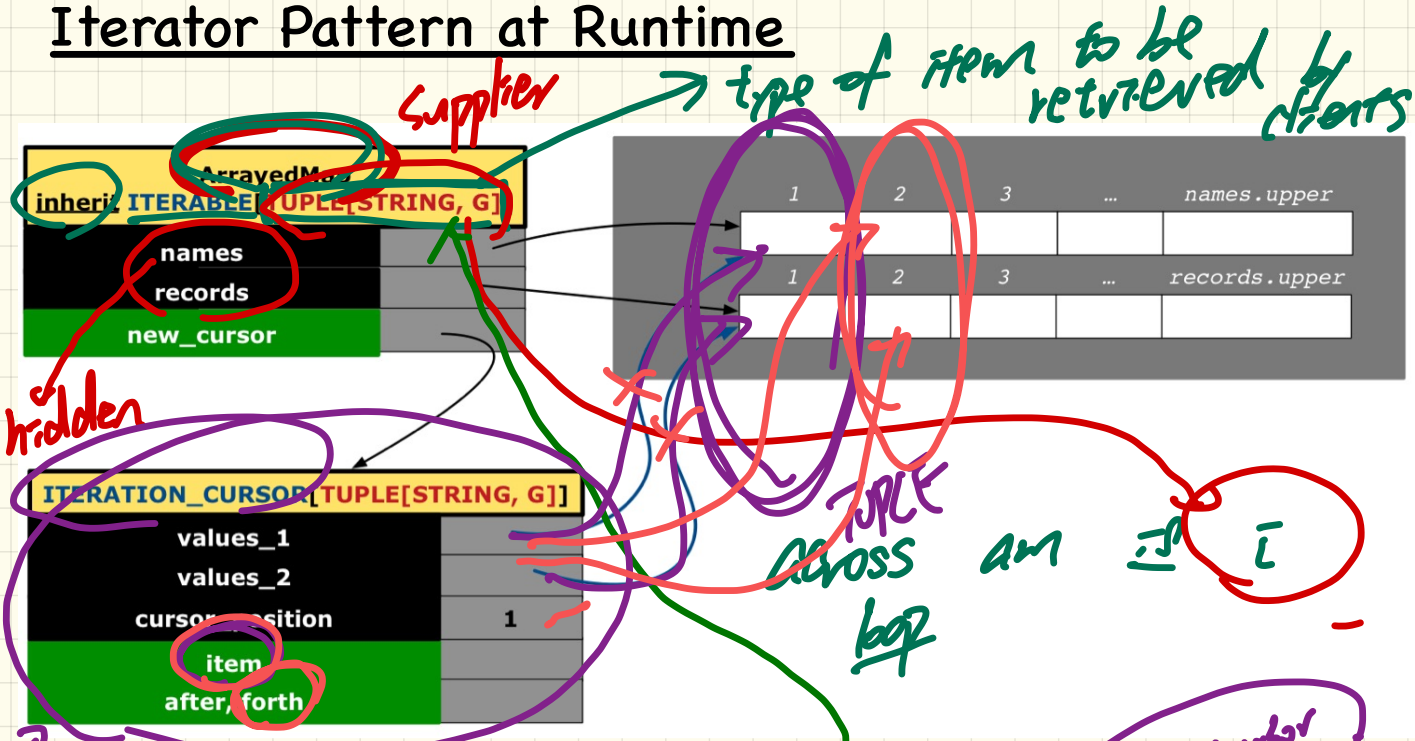
Implementing the Iterator Pattern: Hard Case



Implementing the Iterator Pattern: Hard Case



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
```

INTERFACE TYPE

Dynamic type of collection can be any descendant class of ITER.

collection: ARRAY[INTEGER]

```
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
```

deferred IT. LL, descendant classes of LIST.

Client.

collection: ITERABLE[STRING]

collection.make_empty

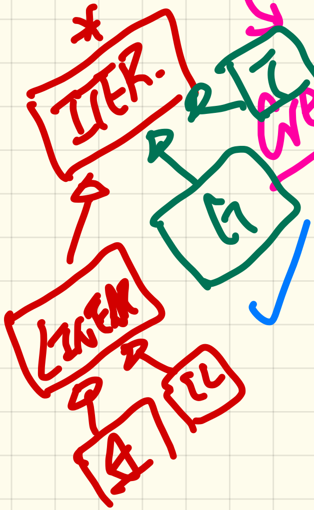
ARRAY[S]

LL[S]

create

collection is STATIC TYPE.

STRING



across collection is i

loop

end

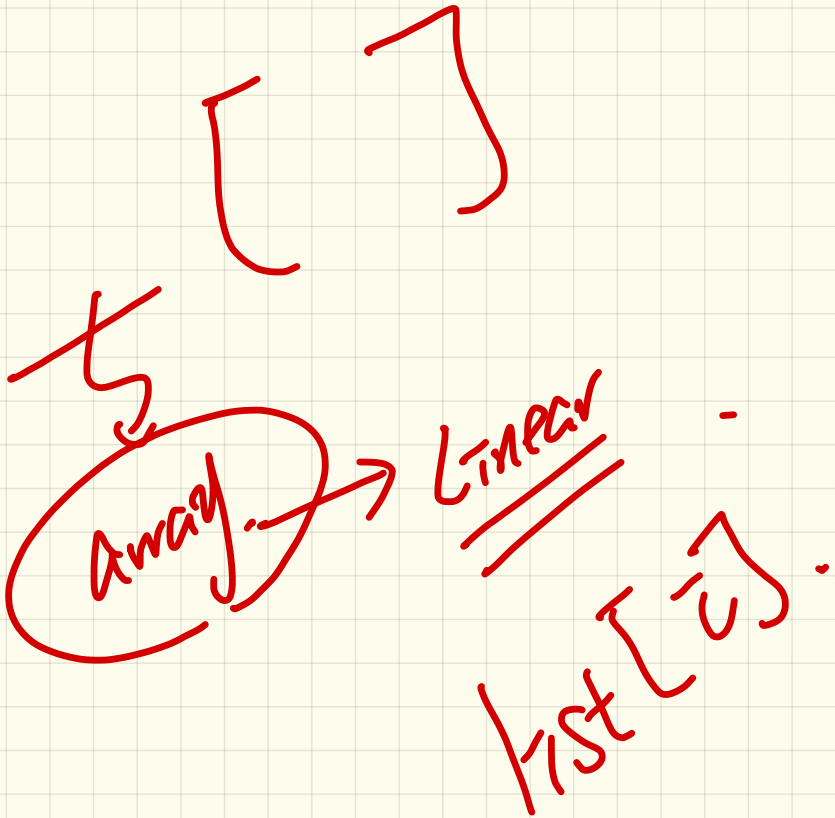
across 1 |..| collection.count is i

loop

collection[i]

collection.item(i)

end


[ ]
array
Linear
list [ ].

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
```

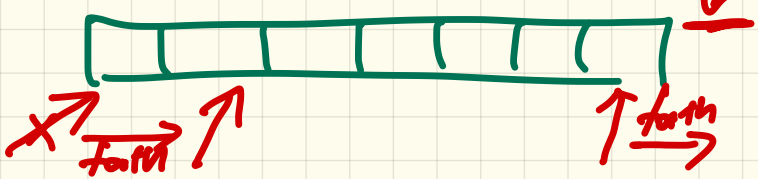
cannot be ITERABLE ∵ we want to refer to positions.

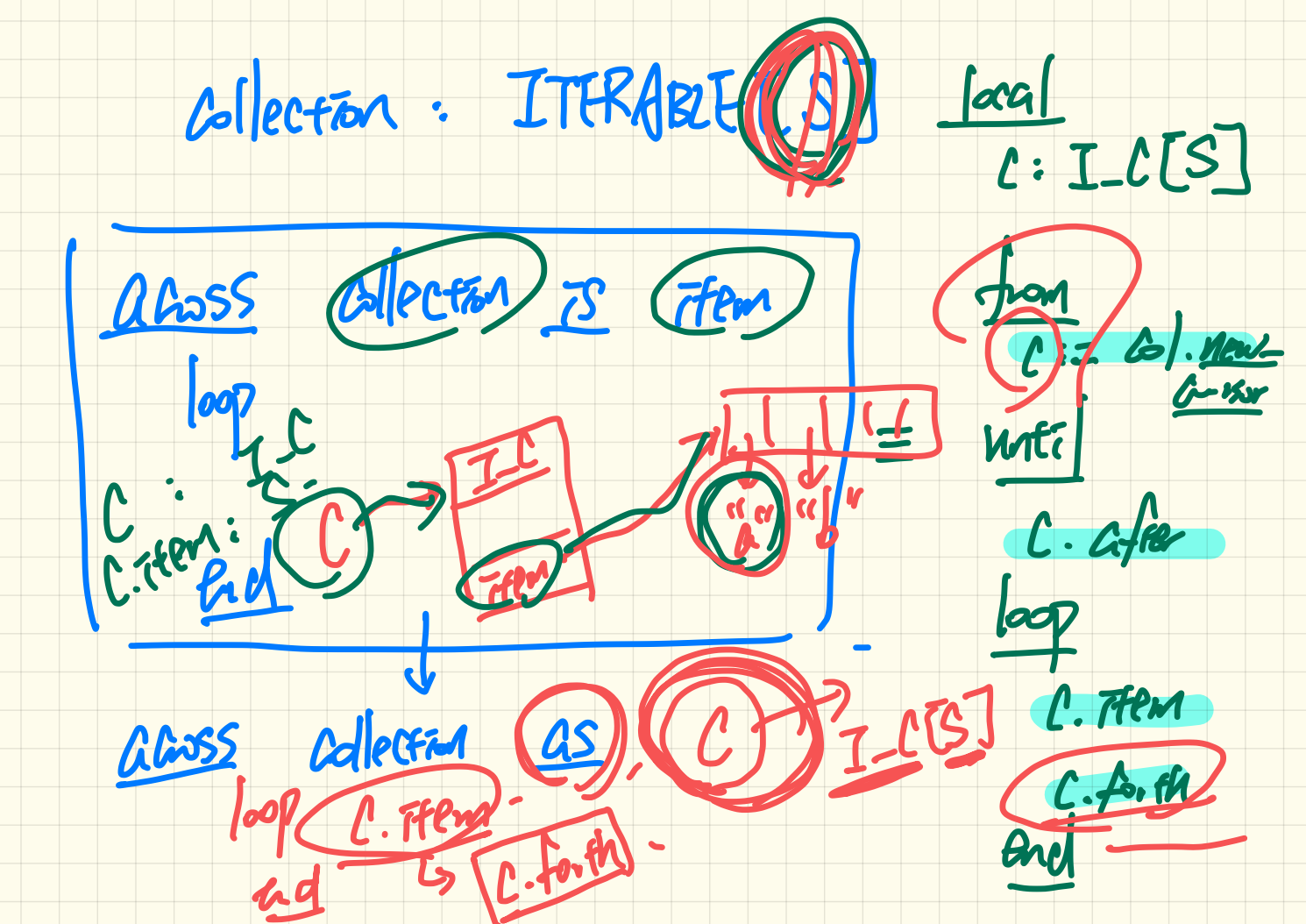
each across can only be bound to a single dum. var.

across | 1..| accounts.count is i, j

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
```




Collection : ITERABLE [S]
local
c : I_C[S]
class Collection is item
loop
c
c : c.item
c.forth
I_C
item
from
c := Col.new_cursor
until
c.after
loop
c.item
c.forth
end
class Collection as c
I_C[S]
loop
c.item
c.forth
end

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
```

db+

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

new_cursor+

```
class
  ITEM_ITERATION_CURSOR[M, N]
inherit
  ITERATION_CURSOR[                ]
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:
    do ... end

  after: BOOLEAN
    do ... end

  forth
    do ... end
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
```

+db

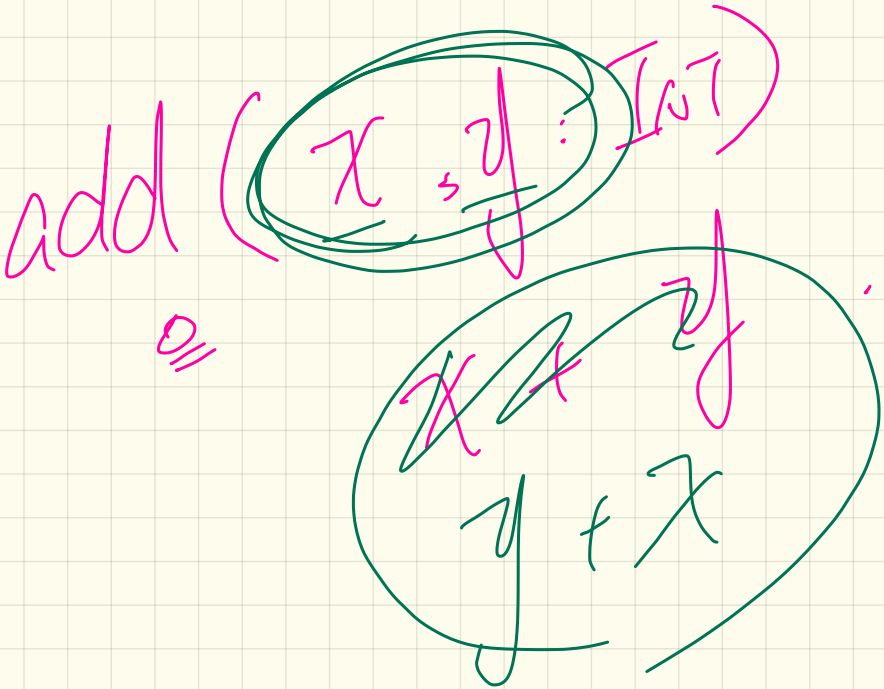
```
class
  DATABASE[G, H]
inherit
  ITERABLE[TUPLE[H, G]]
feature {NONE} -- Implementation
  gs: ARRAY[G]
  hs: ARRAY[H]
feature -- Iterable
  new_cursor: ITERATION_CURSOR[TUPLE[H, G]]
    local
      db_cursor: ITEM_ITERATION_CURSOR[H, G]
    do
      create db_cursor.make (gs, hs)
      Result :=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

new_cursor+

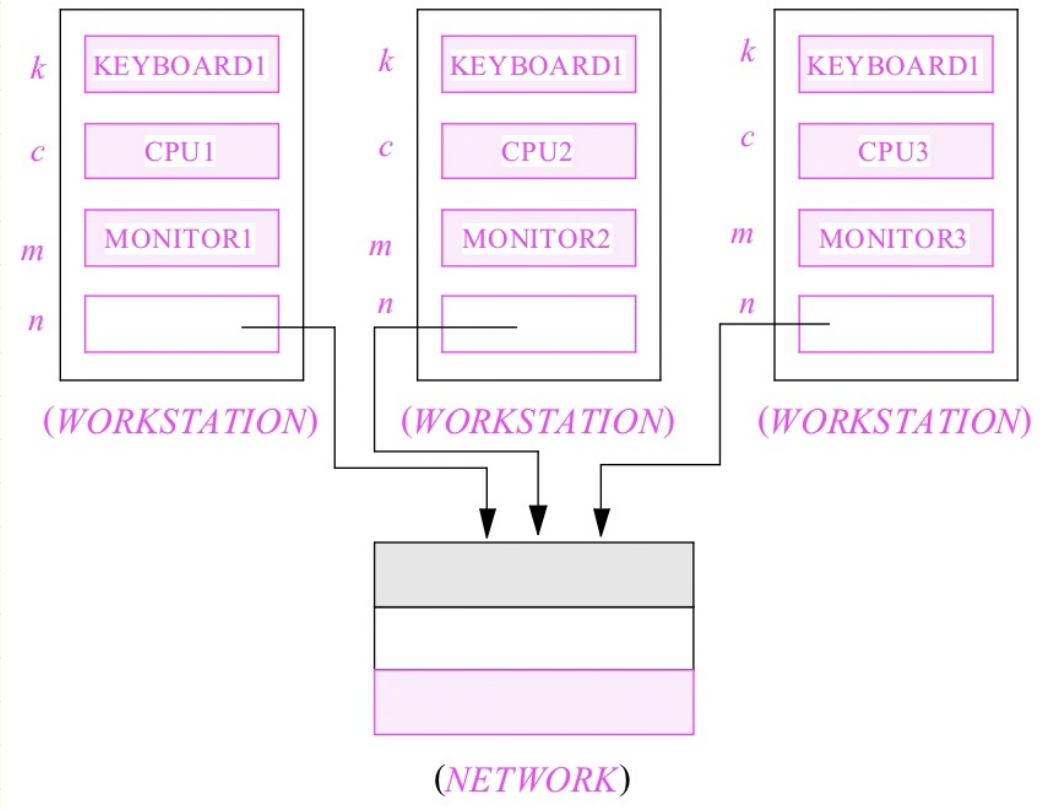
```
class
  ITEM_ITERATION_CURSOR[M, N]
inherit
  ITERATION_CURSOR[TUPLE[M, N]]
create
  make
feature {NONE} -- Implementation
  ms: ARRAY[M]
  ns: ARRAY[N]
feature -- Constructor
  make (new_ns: ARRAY[N]; new_ms: ARRAY[M])
    do ... end
feature -- Cursor features
  item: TUPLE[M, N]
    do ... end

  after: BOOLEAN
    do ... end

  forth
    do ... end
end
```


add
x, y : Int
x ≠ y
y ≠ x

Modelling: Aggregation vs. Composition



Expanded Type for Composition

```
class KEYBOARD ... end class CPU ... end
class MONITOR ... end class NETWORK ... end
class WORKSTATION
  k: expanded KEYBOARD
  c: expanded CPU
  m: expanded MONITOR
  n: NETWORK
end
```

```
expanded class KEYBOARD ... end
expanded class CPU ... end
expanded class MONITOR ... end
class NETWORK ... end
class WORKSTATION
  k: KEYBOARD
  c: CPU
  m: MONITOR
  n: NETWORK
end
```

change: monitor may be shared