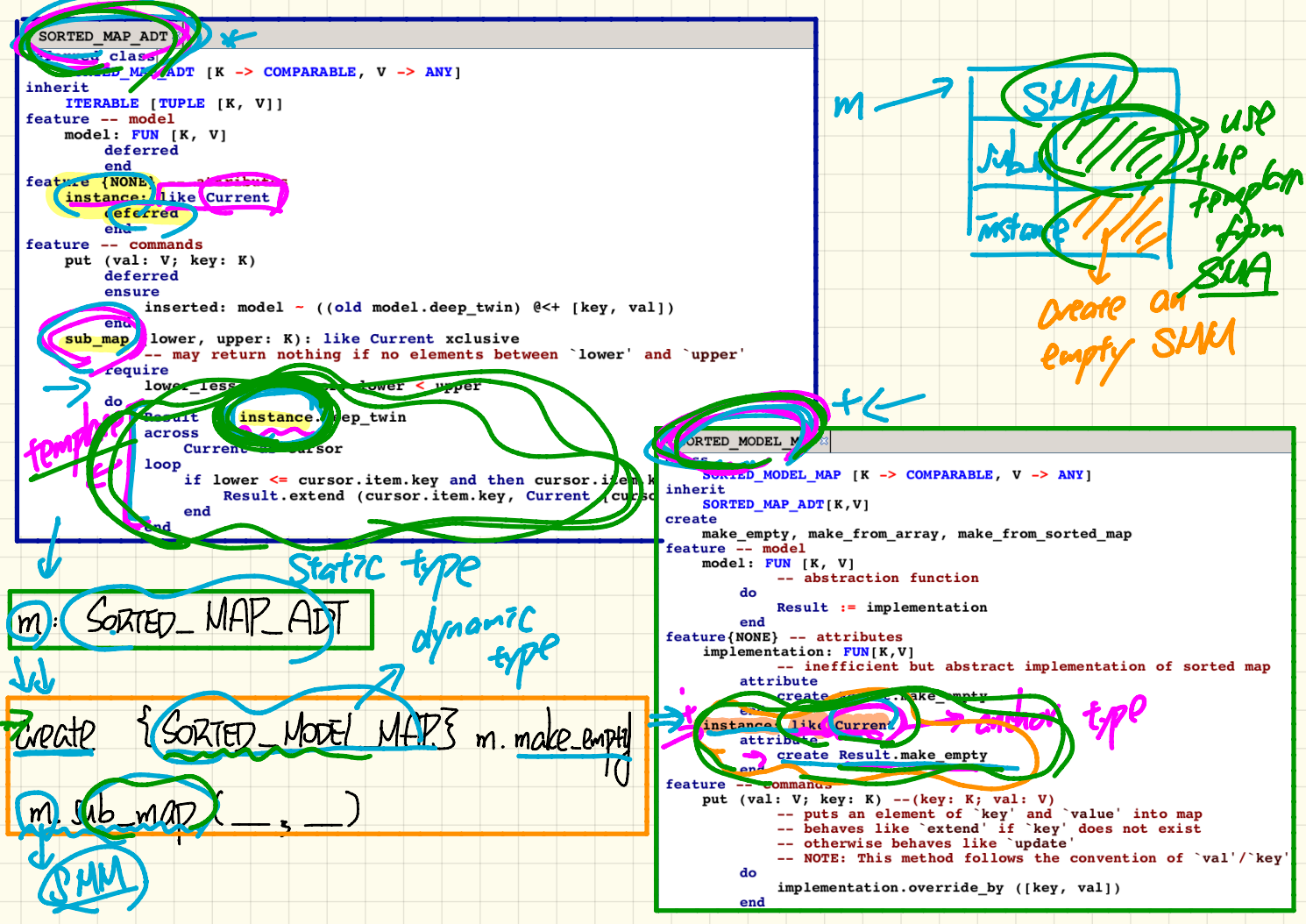
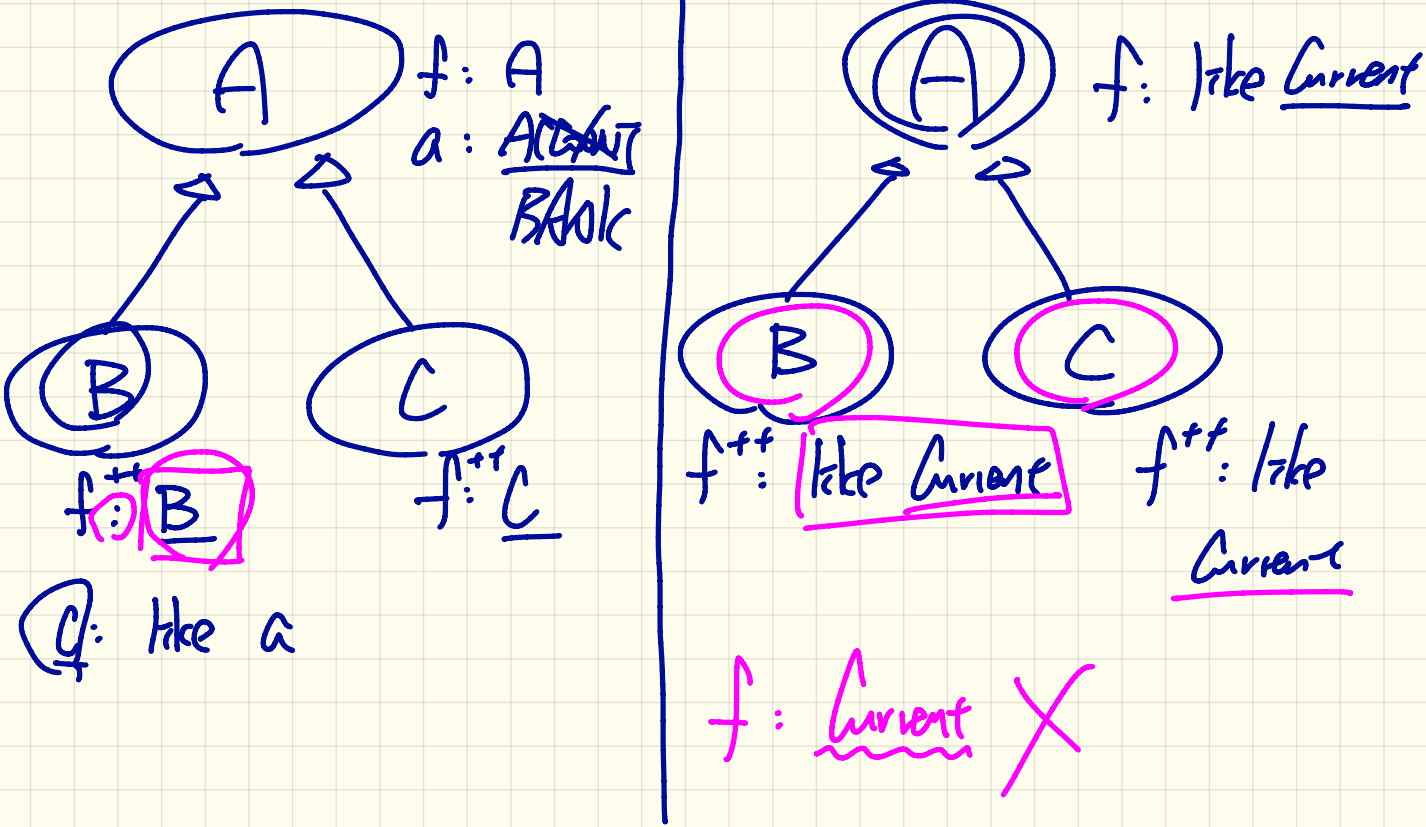
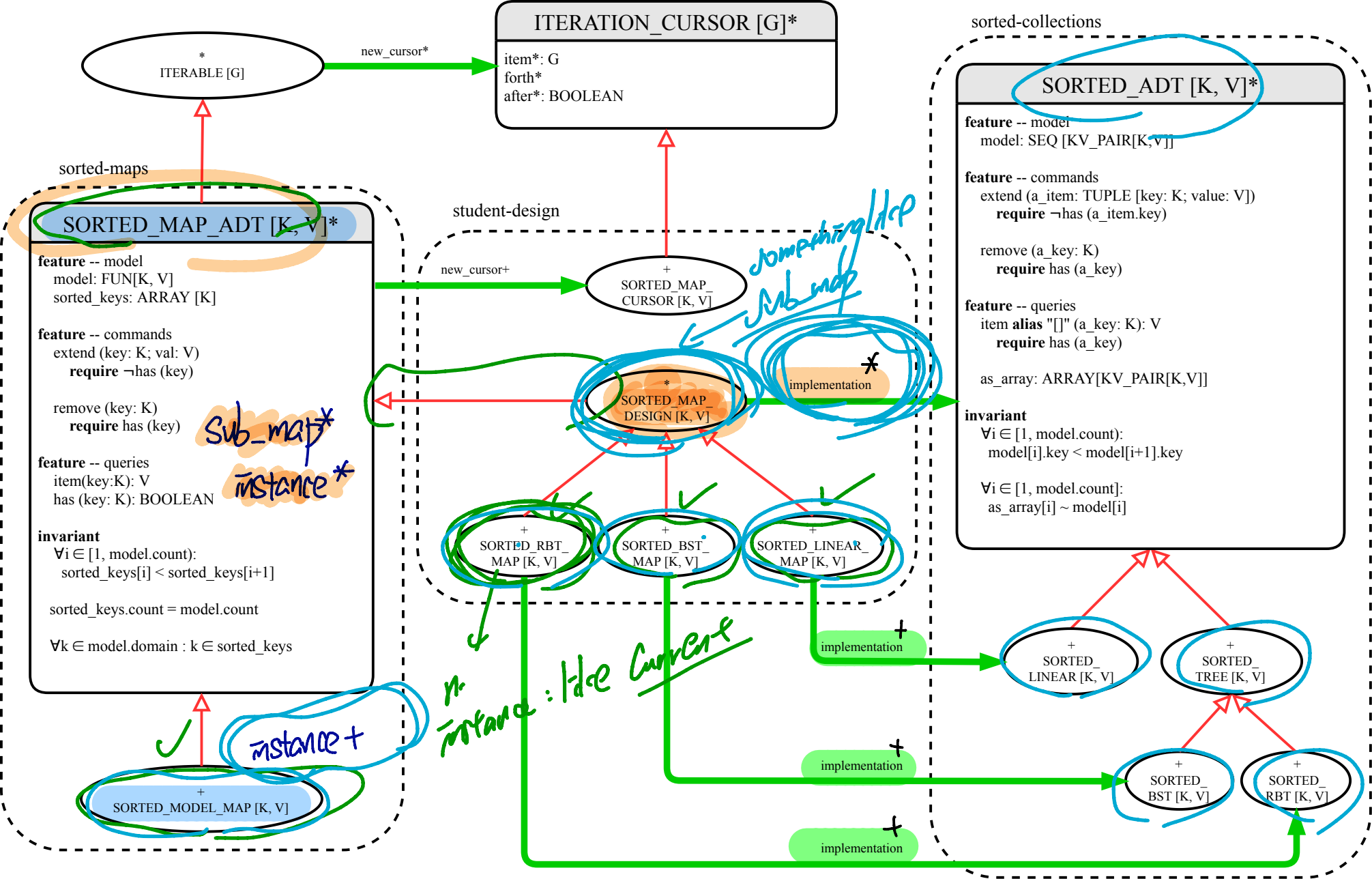
Wednesday January 30

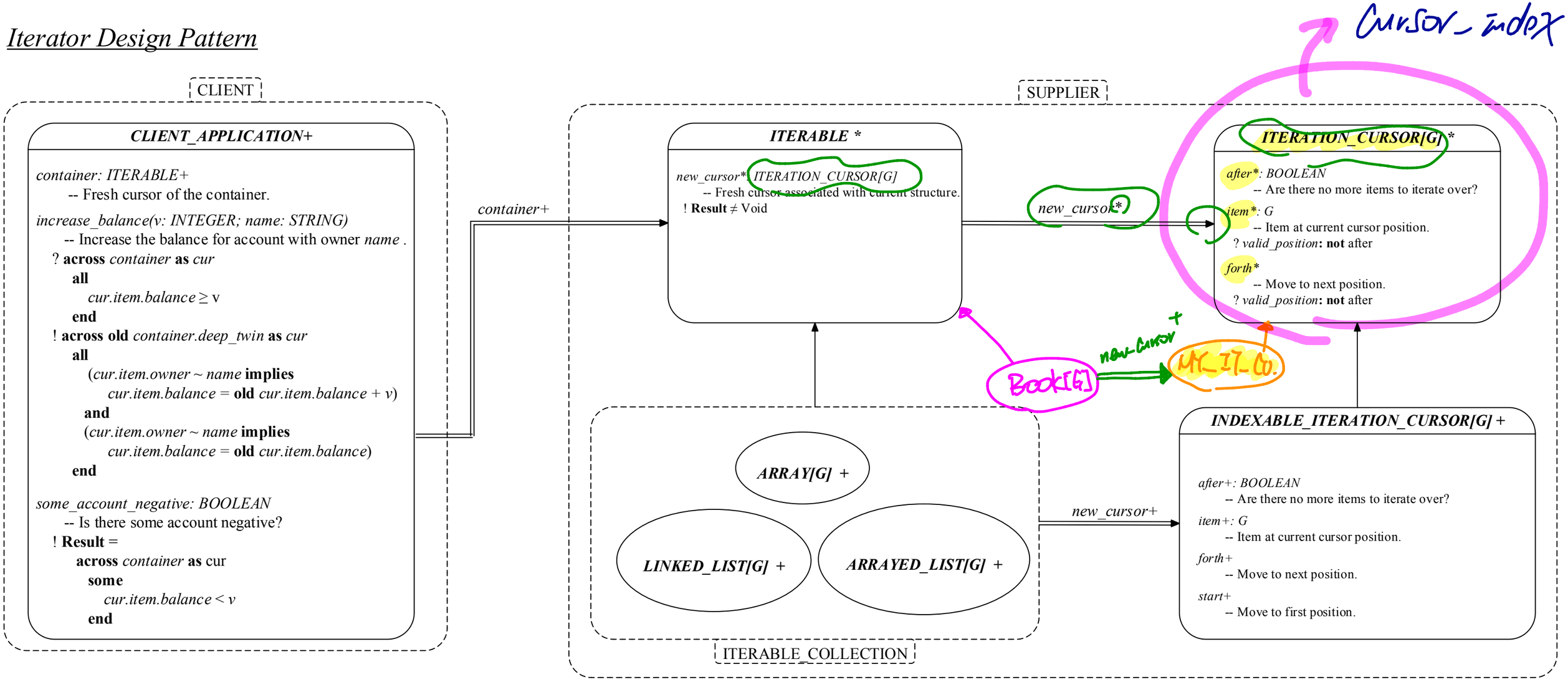
Lecture 8


SORTED_MAP_ADT
class
SORTED_MAP_ADT [K -> COMPARABLE, V -> ANY]
inherit
ITERABLE [TUPLE [K, V]]
feature -- model
model: FUN [K, V]
deferred
end
feature {NONE}
instance: like Current
deferred
end
feature -- commands
put (val: V; key: K)
deferred
ensure
inserted: model ~ ((old model.deep_twin) @<+ [key, val])
end
sub_map (lower, upper: K): like Current xclusive
-- may return nothing if no elements between `lower' and `upper'
require
do
Result := instance.deep_twin
across
Current
loop
if lower <= cursor.item.key and then cursor.item.k
Result.extend (cursor.item.key, Current
end
end
template
m
SUM
use the template from SUM
create an empty SUM
SORTED_MODEL_MAP
SORTED_MODEL_MAP [K -> COMPARABLE, V -> ANY]
inherit
SORTED_MAP_ADT[K,V]
create
make_empty, make_from_array, make_from_sorted_map
feature -- model
model: FUN [K, V]
-- abstraction function
do
Result := implementation
end
feature{NONE} -- attributes
implementation: FUN[K,V]
-- inefficient but abstract implementation of sorted map
attribute
create
instance: like Current
attribute
create Result.make_empty
end
feature -- commands
put (val: V; key: K) --(key: K; val: V)
-- puts an element of `key' and `value' into map
-- behaves like `extend' if `key' does not exist
-- otherwise behaves like `update'
-- NOTE: This method follows the convention of `val'/`key'
do
implementation.override_by ([key, val])
end
m : SORTED_MAP_ADT
static type
dynamic type
create {SORTED_MODEL_MAP} m.make_empty
m.sub_map (__, __)
SUM

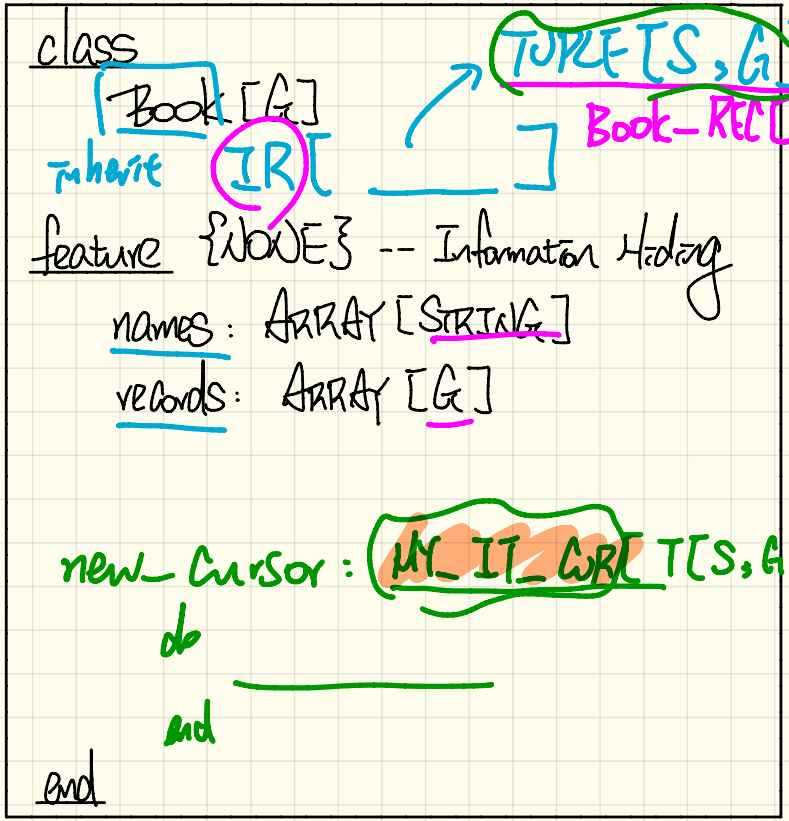

A
f: A
a: ACCOUNT
BASIC
B
C
f++: B
f++: C
g: like a
A
f: like Current
B
C
f++: like Current
f++: like Current
f: Current X


ITERABLE [G]
new_cursor*
ITERATION_CURSOR [G]*
item*: G
forth*
after*: BOOLEAN
sorted-collections
SORTED_ADT [K, V]*
feature -- model
model: SEQ [KV_PAIR[K,V]]
feature -- commands
extend (a_item: TUPLE [key: K; value: V])
require ¬has (a_item.key)
remove (a_key: K)
require has (a_key)
feature -- queries
item alias "[]" (a_key: K): V
require has (a_key)
as_array: ARRAY[KV_PAIR[K,V]]
invariant
∀i ∈ [1, model.count):
model[i].key < model[i+1].key
∀i ∈ [1, model.count]:
as_array[i] ~ model[i]
sorted-maps
SORTED_MAP_ADT [K, V]*
feature -- model
model: FUN[K, V]
sorted_keys: ARRAY [K]
feature -- commands
extend (key: K; val: V)
require ¬has (key)
remove (key: K)
require has (key)
feature -- queries
item(key:K): V
has (key: K): BOOLEAN
invariant
∀i ∈ [1, model.count):
sorted_keys[i] < sorted_keys[i+1]
sorted_keys.count = model.count
∀k ∈ model.domain : k ∈ sorted_keys
student-design
new_cursor+
+ SORTED_MAP_CURSOR [K, V]
* SORTED_MAP_DESIGN [K, V]
implementation
+ SORTED_RBT_MAP [K, V]
+ SORTED_BST_MAP [K, V]
+ SORTED_LINEAR_MAP [K, V]
+ SORTED_MODEL_MAP [K, V]
implementation
implementation
implementation
+ SORTED_LINEAR [K, V]
+ SORTED_TREE [K, V]
+ SORTED_BST [K, V]
+ SORTED_RBT [K, V]

Iterator Design Pattern



Implementing the ITERATOR Pattern: Hard Case



Book_REC[STRING, G]

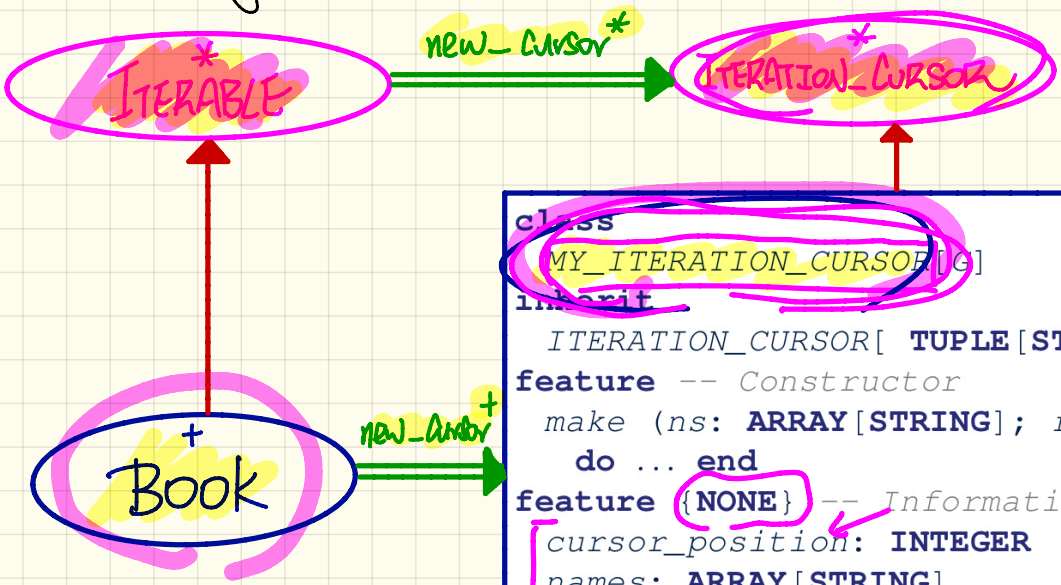
class Book_RECORD[S, G]

name: STRING

record: G

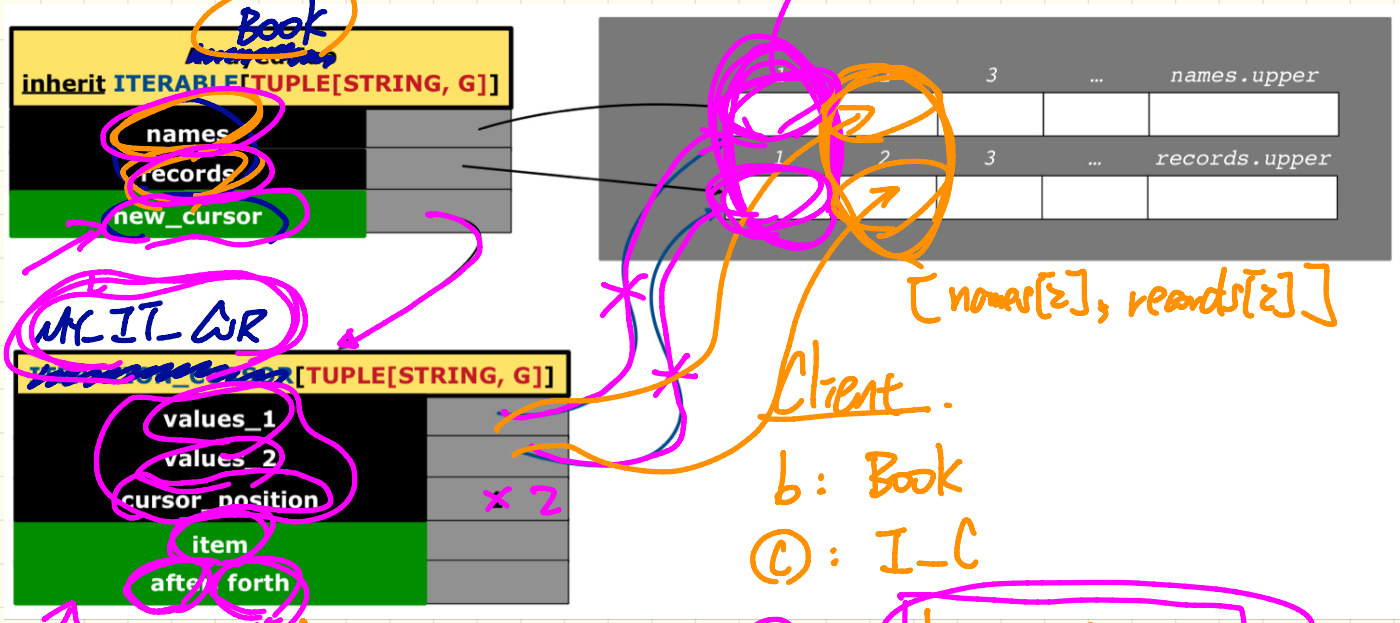
end

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 as cursor
      all
        cursor.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) as cursor
      all
        accounts [cursor.item].id <= accounts [cursor.item + 1].id
      end
    do
      ...
    ensure
      Result.id = acc_id
    end
```

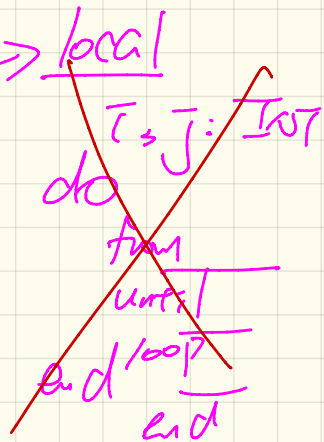
Use of Iterable in Contracts: Exercise

```
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
```

Contra-positive

$p \Rightarrow q \equiv \neg q \Rightarrow \neg p$

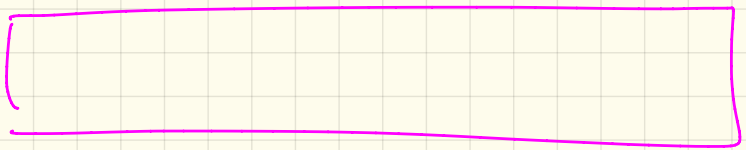
$i \neq j \Rightarrow accounts[i] \not\sim accounts[j]$



across | 1..| accounts.count as i

across [i] 1..| accounts.count as j

i.item



end

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
```

across accounts as cursor loop

max := cursor.item

--no need to say cursor.forth

end

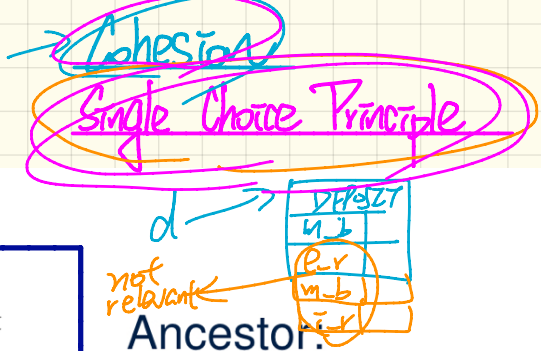
Use of Iterable in Implementation (2)

```
class SHOP
  cart: CART
  checkout: INTEGER
      -- Total price calculated based
    require ??
    local
      order: ORDER
    do
      across
        cart as cursor
      loop
        order := cursor.item
        Result := Result + order.price * order.quantity
      end
    ensure ??
  end
```

no cursor.forth

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

Shared Data via Inheritance



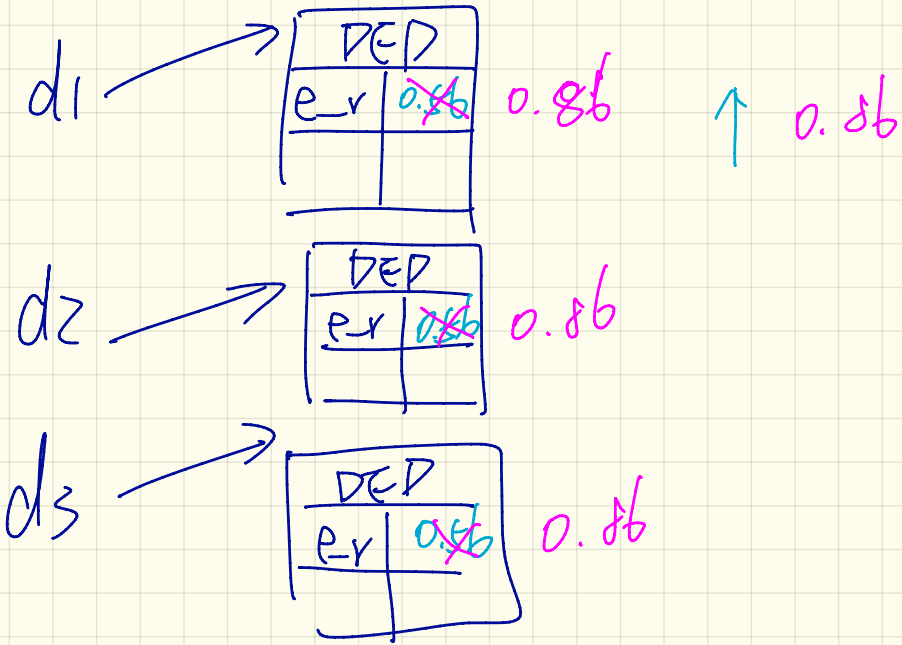
Descendant:

```
class DEPOSIT inherit SHARED_DATA
        -- 'maximum_balance' relevant
end

class WITHDRAW inherit SHARED_DATA
        -- 'minimum_balance' relevant
end

class INT_TRANSFER inherit SHARED_DATA
        -- 'exchange_rate' relevant
end

class ACCOUNT inherit SHARED_DATA
feature
        -- 'interest_rate' relevant
        deposits: DEPOSIT_LIST
        withdraws: WITHDRAW_LIST
end
```

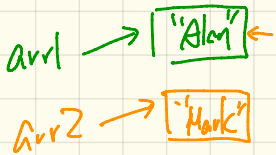
Ancestor:

```
class
  SHARED_DATA
feature
  interest_rate: REAL
  exchange_rate: REAL
  minimum_balance: INTEGER
  maximum_balance: INTEGER
  ...
end
```

Problems?


d1
DED
e_r
0.8
d2
DED
e_r
0.8
d3
DED
e_r
0.8

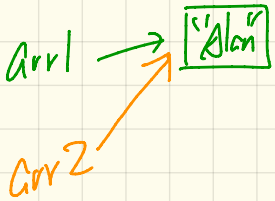
Once Routine (1)




```
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
```

```
class A
create make
feature -- Constructor
  make do end
feature -- 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
  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
end
```

Once Routine (2)




```
class A
create make
feature -- Constructor
  make do end
feature -- 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
  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
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
```