Lecture 6

Tuesday September 24

- Lab Test 1

Guide

Practice Questions

- Lab 2

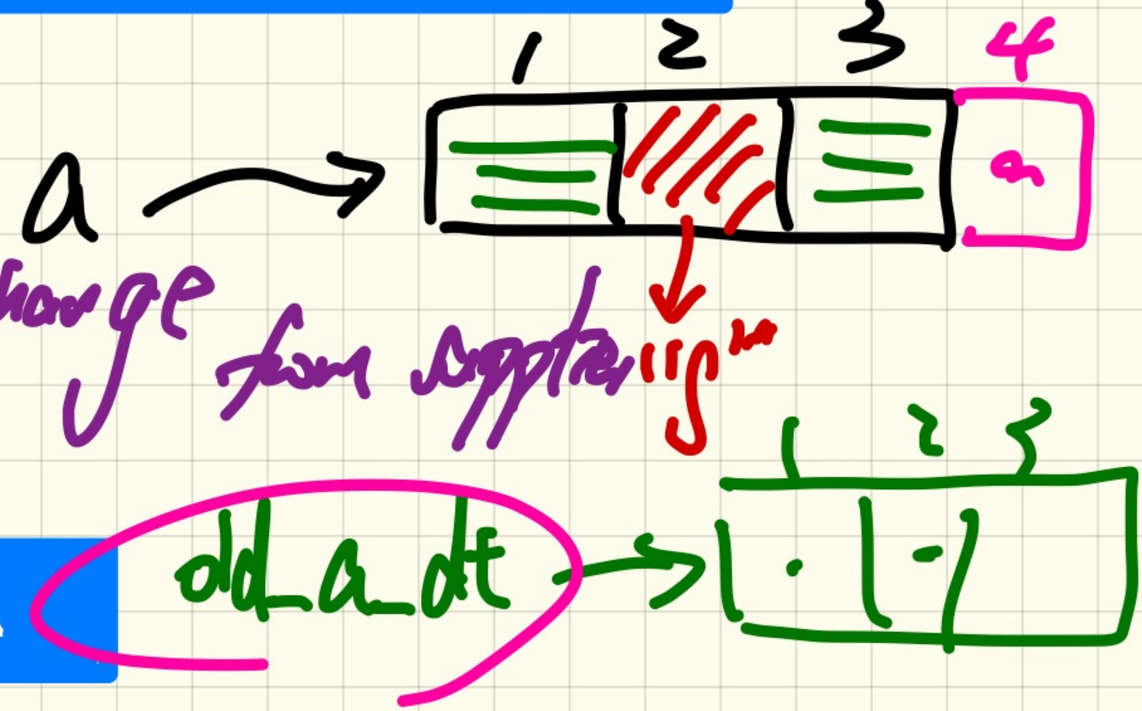
Writing Postcondition: Exercise 2

```
a: ARRAY[STRING]
change_at (i: INTEGER; s: STRING)
  ensure
    across a.lower |..| a.upper is j
      all
        j = i implies a[j] ~ s
        and
        j /= i implies a[j] ~ old a.deep_twin[j]
    end
```

Complete Postcondition?

What if a.count > **old** a.count?

What if a.count < **old** a.count?



Writing Postcondition: Exercise 3

Pos: positive numbers in a

```
all_positive_values (a: ARRAY[INTEGER]): ARRAY[INTEGER]
    require
        no_duplicates: ??
    ensure
        across Result is x
            all
                x > 0
            end
```

a ~ old a.deep_twin

Pos ⊆ Result

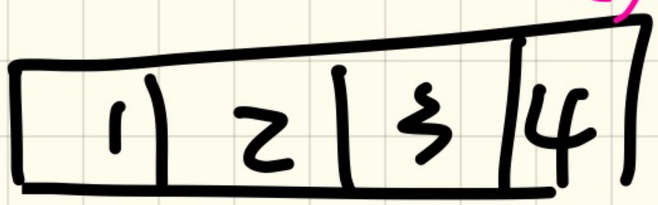
Result ⊆ Pos

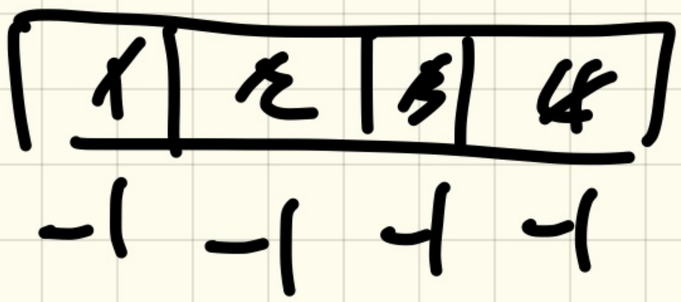
① $\forall x \mid x \in Result \mid x > 0 \wedge x \in a$

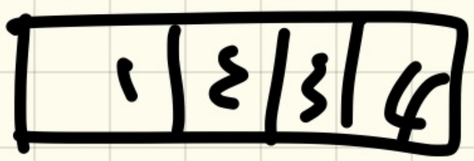
a. # of pos #s in a = Result.count

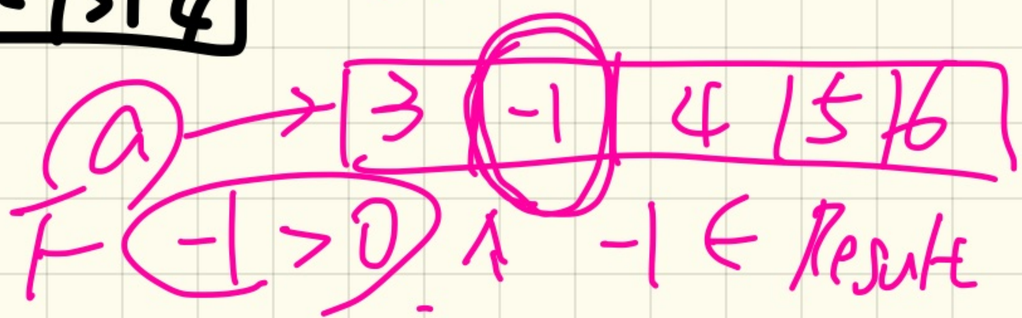
② b. $\forall x \mid x \in a \mid x > 0 \Rightarrow$ ③ $x \in Result$

FTCTP

① 

② 





1. How the collection works

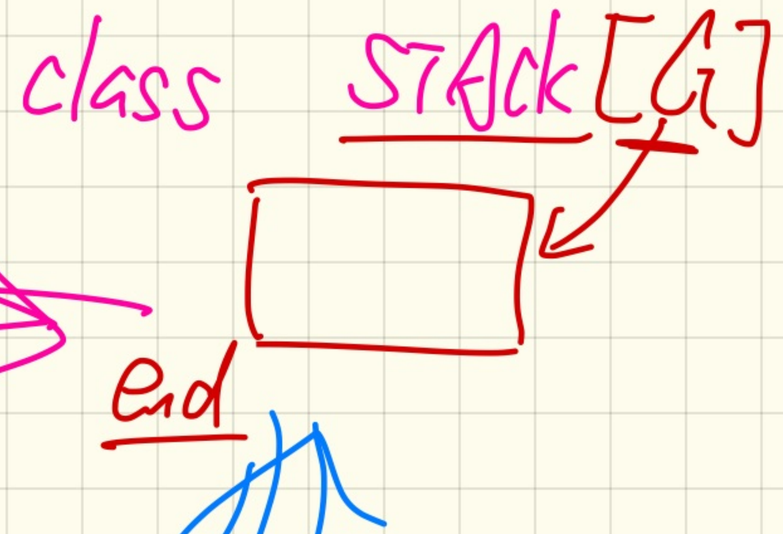
LIFO

2. type of collection members

(STRING)

Stack of Strings vs. Stack of Accounts

```
class STRING_STACK
feature {NONE} -- Implementation
  imp: ARRAY[STRING] ; i: INTEGER
feature -- Queries
  count: INTEGER do Result := i end
     -- Number of items on stack.
  top: STRING do Result := imp [i] end
     -- Return top of stack.
feature -- Commands
  push (v: STRING) do imp[i] := v; i := i + 1 end
     -- Add 'v' to top of stack.
  pop do i := i - 1 end
     -- Remove top of stack.
end
```




```
class ACCOUNT_STACK
feature {NONE} -- Implementation
  imp: ARRAY[ACCOUNT] ; i: INTEGER
feature -- Queries
  count: INTEGER do Result := i end
     -- Number of items on stack.
  top: ACCOUNT do Result := imp [i] end
     -- Return top of stack.
feature -- Commands
  push (v: ACCOUNT) do imp[i] := v; i := i + 1 end
     -- Add 'v' to top of stack.
  pop do i := i - 1 end
     -- Remove top of stack.
end
```

A Generic Stack

Supplier

```
class STACK [G]
feature {NONE} -- Implementation
  imp: ARRAY [G]; i: INTEGER
feature -- Queries
  count: INTEGER do Result := i end
      -- Number of items on stack.
  top: G do Result := imp [i] end
      -- Return top of stack.
feature -- Commands
  push (v: G) do imp[i] := v; i := i + 1 end
      -- Add 'v' to top of stack.
  pop do i := i - 1 end
      -- Remove top of stack.
end
```

Client

```
1  test_stacks: BOOLEAN
2   local
3    ss: STACK[STRING] ; sa: STACK[ACCOUNT]
4    s: STRING ; a: ACCOUNT
5   do
6    ss.push("A")
7    ss.push(create {ACCOUNT}.make ("Mark", 200))
8    s := ss.top
9    a := ss.top
10   sa.push(create {ACCOUNT}.make ("Alan", 100))
11   sa.push("B")
12   a := sa.top
13   s := sa.top
14  end
```

Java

```
class Collection < E extends Comparable >
```

Eiffel

```
C : Collection < ? >
```

? ? any descendant of class of Comparable.

```
class COLLECTION [ E -> COMPARABLE ]
```

Principle of Information Hiding

Supplier:

```
class
  CART
feature
  orders: ARRAY[ORDER]
end

class
  ORDER
feature
  price: INTEGER
  quantity: INTEGER
end
```

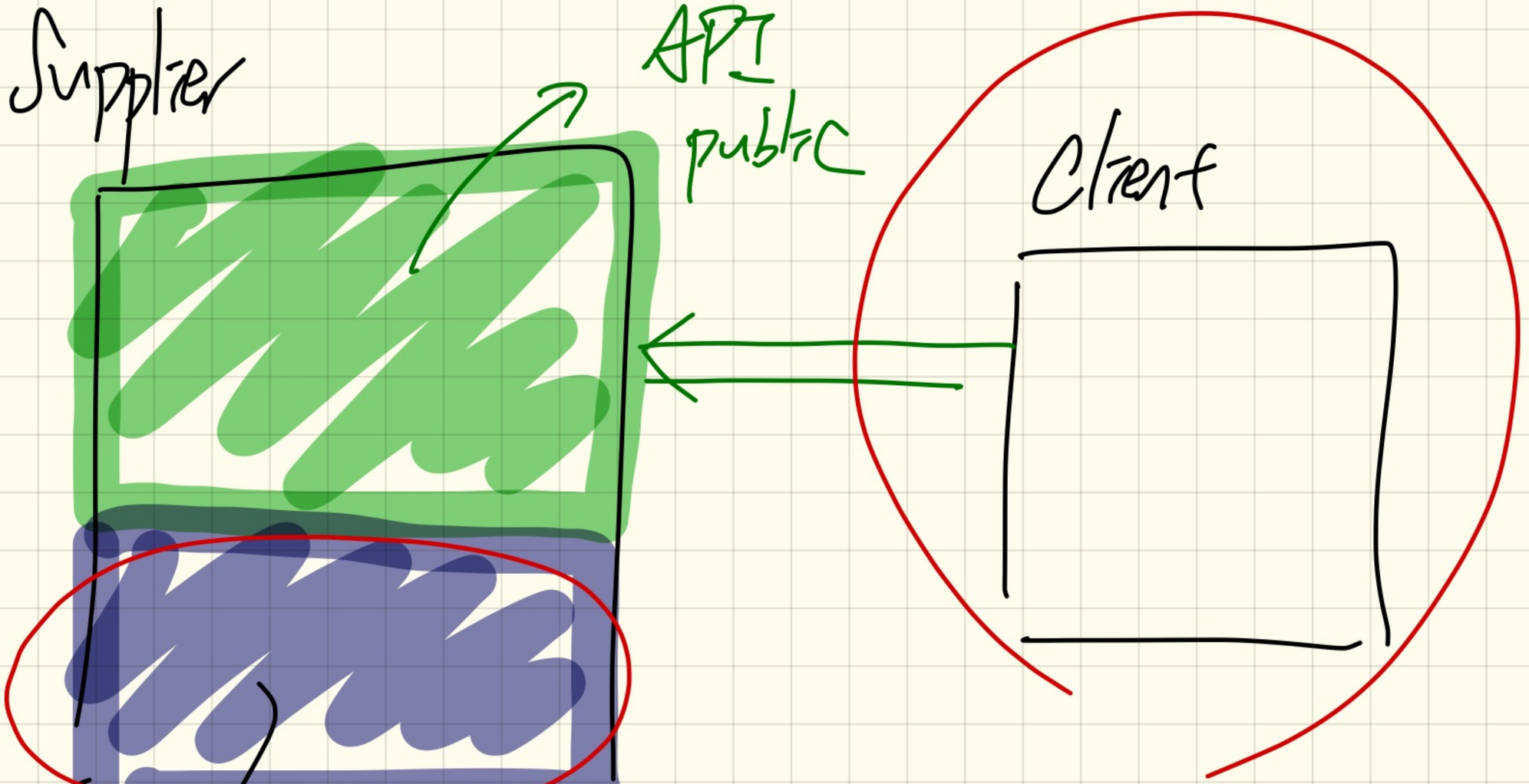
Problems?

Client:

```
class
  SHOP
feature
  cart: CART
  checkout: INTEGER
    do
      from
        i := cart.orders.lower
      until
        i > cart.orders.upper
      do
        Result := Result +
          cart.orders[i].price
          *
          cart.orders[i].quantity
        i := i + 1
      end
    end
end
```

1. As supplier, you should be free to change imp. strategy!

2. When you change imp. strategy, all existing clients should NOT be affected



design secrete that's
hidden.

Iterator Pattern at Runtime

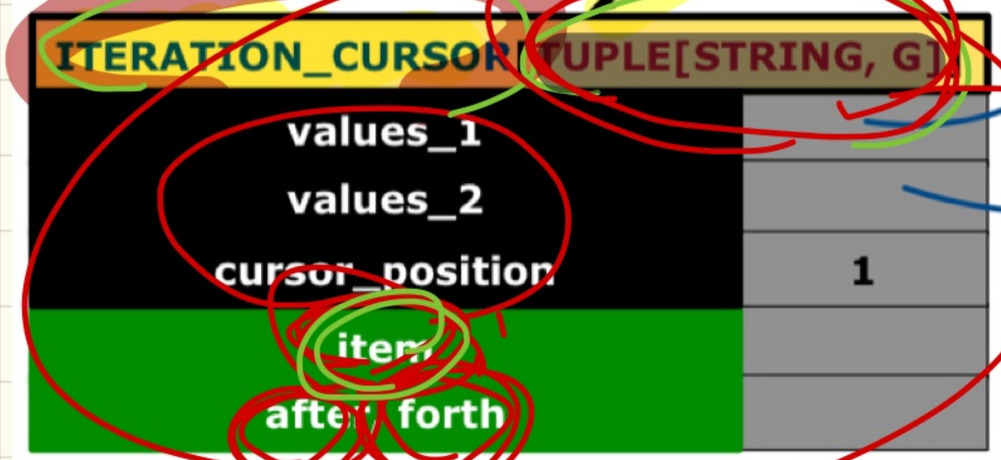
feature {NONE} item's type?

App

when retrieving an item from A.M. → what should be the

Any client of A.M. cannot access these

??



["Alan", 2]

["mark", 10]

new_cursor is the only way for iterating over the hidden structure

client

```
am : ARRAC_MAP
from
    cursor := am.new_cursor
until
    cursor.after
loop
    print (cursor.item)
    cursor.forth
end
```

Implementing the Iterator Pattern: Easy Case

class
CART
inherit
ITERABLE[ORDER]
feature {**NONE**}
orders: **ARRAY**[ORDER]

feature -- Cursor

new_cursor : ITERATION_CURSOR[ORDER]
do
end Result := orders.new_cursor
end