Tuesday Sep. 25

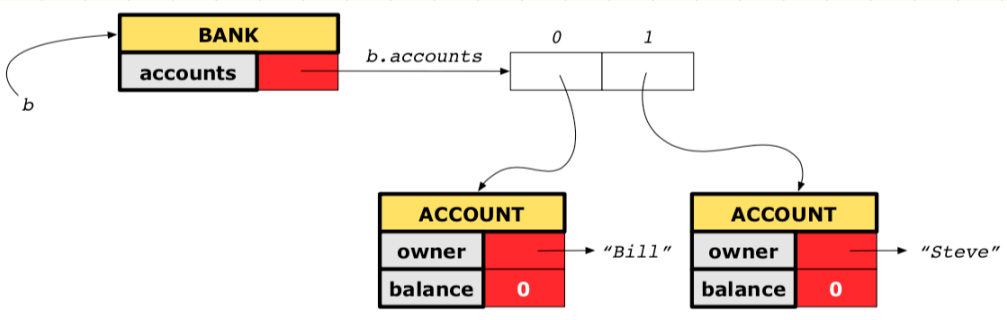
Lecture 6

# MATRIX

TS_equal ( other: ~~MATRIX~~ like Current ) : BOOLEAN

# Version 1 : Incomplete Contracts, Correct Implementation

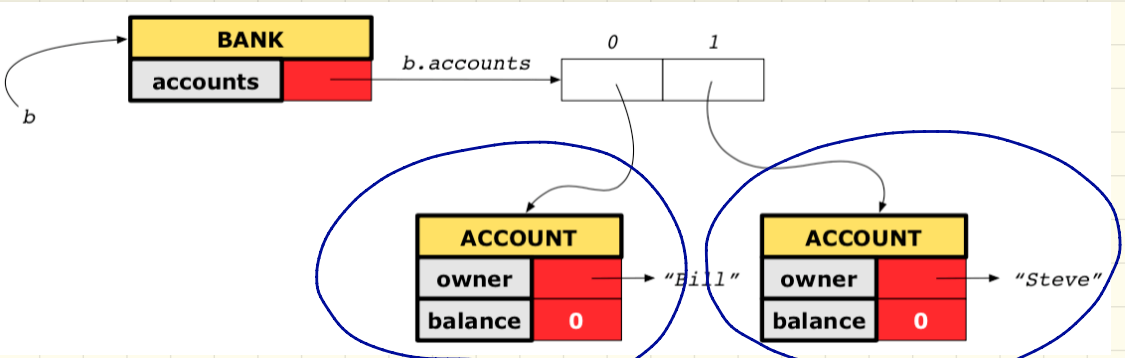
b.deposit ("Steve", 100)




```
class BANK
  deposit_on_v1 (n: STRING; a: INTEGER)
    require across accounts as acc some acc.item.owner ~ n end
    local i: INTEGER
    do
      from i := accounts.lower
      until i > accounts.upper
      loop
        if accounts[i].owner ~ n then accounts[i].deposit(a) end
        i := i + 1
      end
    ensure
      num_of_accounts_unchanged:
        accounts.count = old accounts.count
      balance_of_n_increased:
        account_of (n).balance = old account_of (n).balance + a
    end
end
```

# Version 2 : Incomplete Contracts, Wrong Implementation

b.deposit ("Steve", 100)

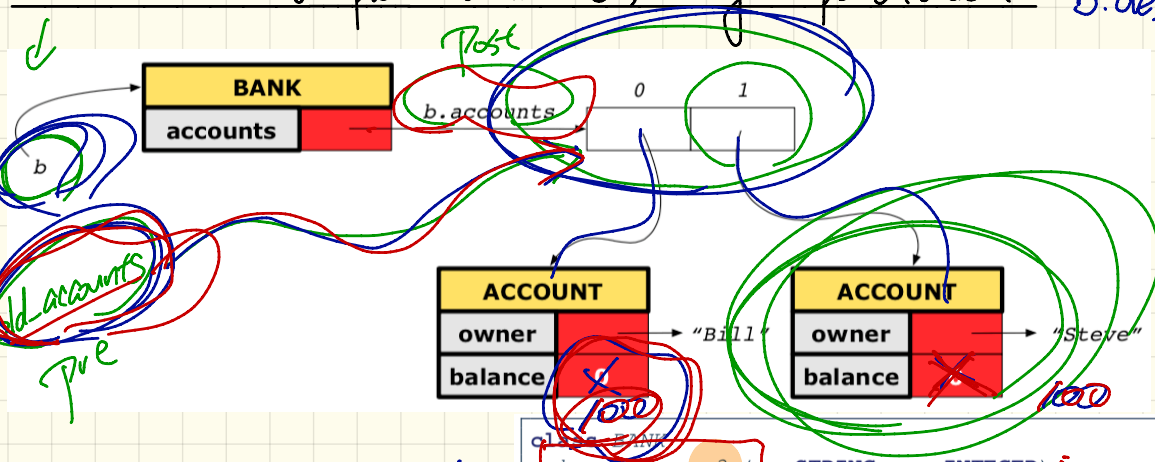



```
class BANK
  deposit_on_v2 (n: STRING; a: INTEGER)
    require across accounts as acc some acc.item.owner ~ n end
    local i: INTEGER
    do
      -- same loop as in version 1

      -- wrong implementation: also deposit in the first account
      accounts[accounts.lower].deposit(a)
    ensure
      num_of_accounts_unchanged:
        accounts.count = old accounts.count
      balance_of_n_increased:
        account_of (n).balance = old account_of (n).balance + a
    end
end
```

# Version 3: Complete Contracts, Wrong Implementation (Reference Copy)

b.deposit("Steve", 100)



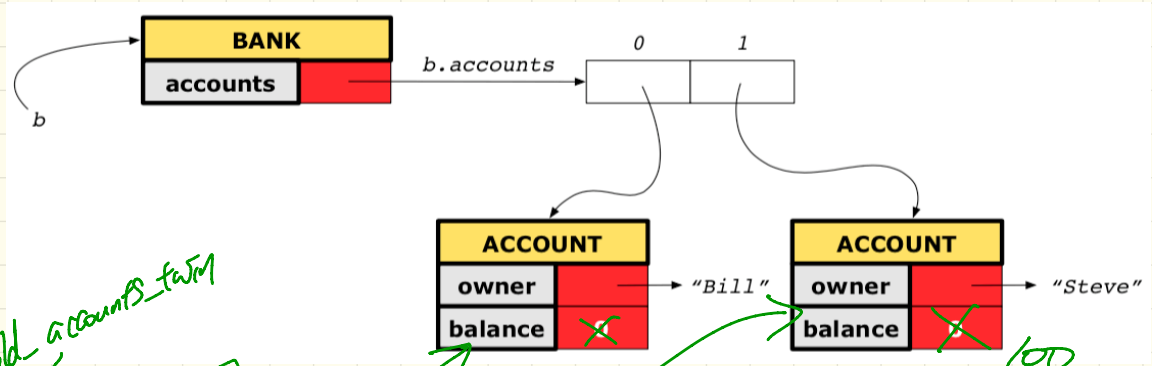

across 1..1 old (accounts.count - 1) as i all old accounts[i.item] ~ account_of(n) end

≡ True

```
class BANK
  deposit_on_v3 (n: STRING; a: INTEGER)
    require across accounts as acc some acc.item.owner ~ n end
    local i: INTEGER
    do
      -- same loop as in version 1
      -- wrong implementation: also deposit in the first account
      accounts[accounts.lower].deposit(a)
    ensure
      num_of_accounts_unchanged: accounts.count = old accounts.count
      balance_of_n_increased:
        account_of(n).balance = old account_of(n).balance + a
      others_unchanged:
        across old accounts as cursor
        all cursor.item.owner /~ n implies
          cursor.item ~ account_of (cursor.item.owner)
        end
    end
end
```

# Version 4: Complete Contracts, Wrong Implementation (shallow copy)

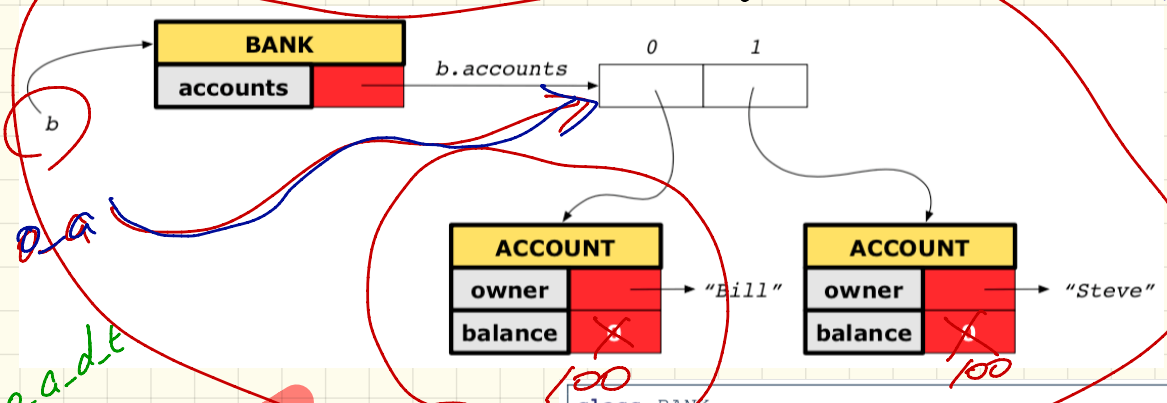
b.deposit ("Steve", 100)




```
class BANK
  deposit_on_v4 (n: STRING; a: INTEGER)
    require across accounts as acc some acc.item.owner ~ n end
    local i: INTEGER
    do
      -- same loop as in version 1
      -- wrong implementation: also deposit in the first account
      accounts[accounts.lower].deposit(a)
    ensure
      num_of_accounts_unchanged: accounts.count = old accounts.count
      balance_of_n_increased:
        account_of (n).balance = old account_of (n).balance + a
      others_unchanged :
        across old accounts.twin as cursor
        all cursor.item.owner /~ n implies
            cursor.item ~ account_of (cursor.item.owner)
      end
    end
end
```

# Version 5: Complete Contracts, Wrong Implementation (Deep Copy)

b.deposit ("Steve", 100)



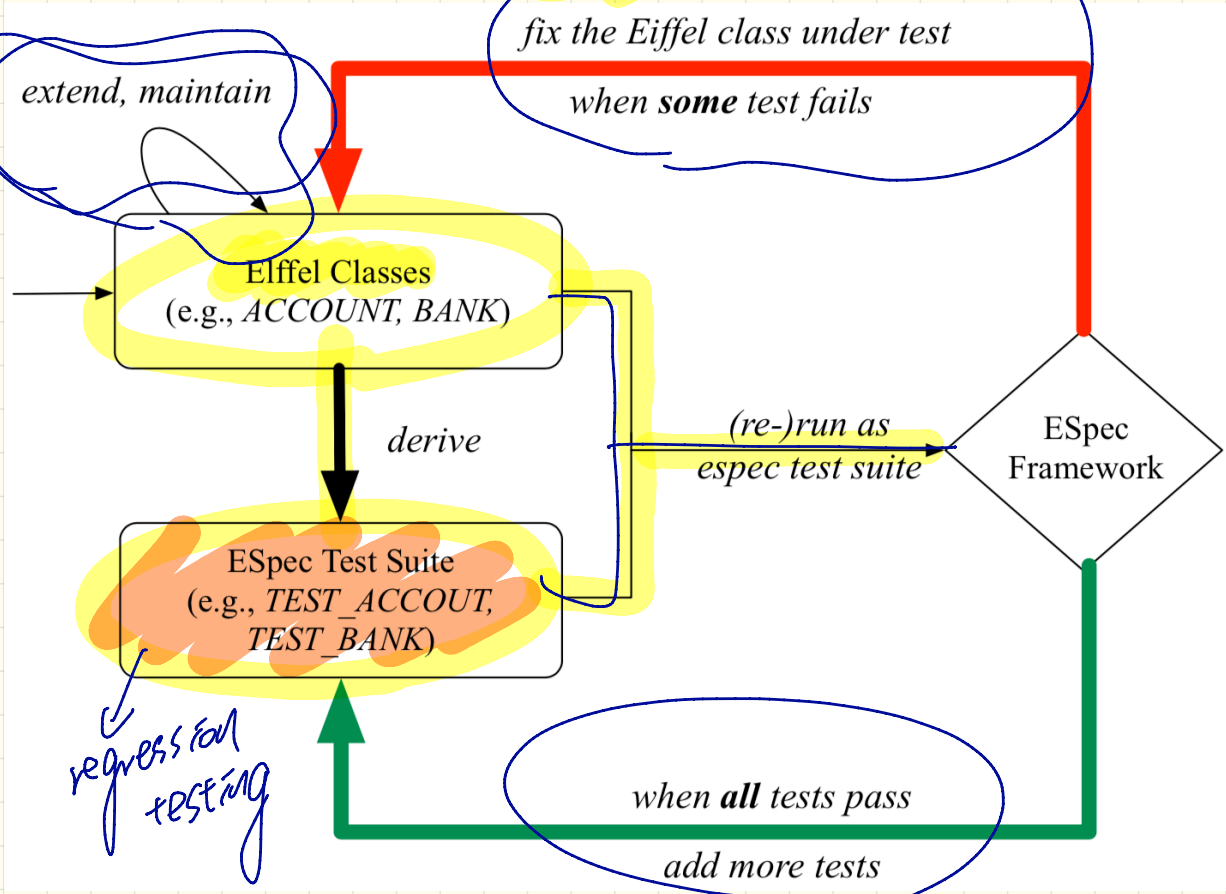

a_a

a_a_d_t

old accounts.deep_twin

```
class BANK
  deposit_on_v5 (n: STRING; a: INTEGER)
    require across accounts as acc some acc.item.owner ~ n end
    local i: INTEGER
  do
    -- same loop as in version 1
    -- wrong implementation: also deposit in the first account
    accounts[accounts.lower].deposit(a)
  ensure
    num_of_accounts_unchanged: accounts.count = old accounts.count
    balance_of_n_increased:
      account_of (n).balance = old account_of (n).balance + a
    others_unchanged :
      across old accounts.deep_twin as cursor
      all cursor.item.owner /~ n implies
          cursor.item ~ account_of (cursor.item.owner)
    end
  end
end
```

# Test-Driven Development (TDD)



$$\forall x \cdot P(x) \equiv \neg(\exists x : \neg P(x))$$

$$\exists x \cdot P(x) \equiv \neg(\forall x : \neg P(x))$$

student

$x$

$P(x)$

get A+

student

does not get A+

# 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
```

STACK[G]

```
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

```
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
```

What happens if the client declares:

cl

s_s: STACK[STRING]

a_s: STACK[ACCOUNT]

# Client of a Generic Stack

```
test_stacks: BOOLEAN
  local
    ss: STACK[STRING] ; sa: STACK[ACCOUNT]
    s: STRING ; a: ACCOUNT
  do
    ss.push("A")
    ss.push(create {ACCOUNT}.make ("Mark", 200))
    s := ss.top
    a := ss.top
    sa.push(create {ACCOUNT}.make ("Alan", 100))
    sa.push("B")
    a := sa.top
    s := sa.top
  end
```

# Information Hiding Principle

## Supplier:

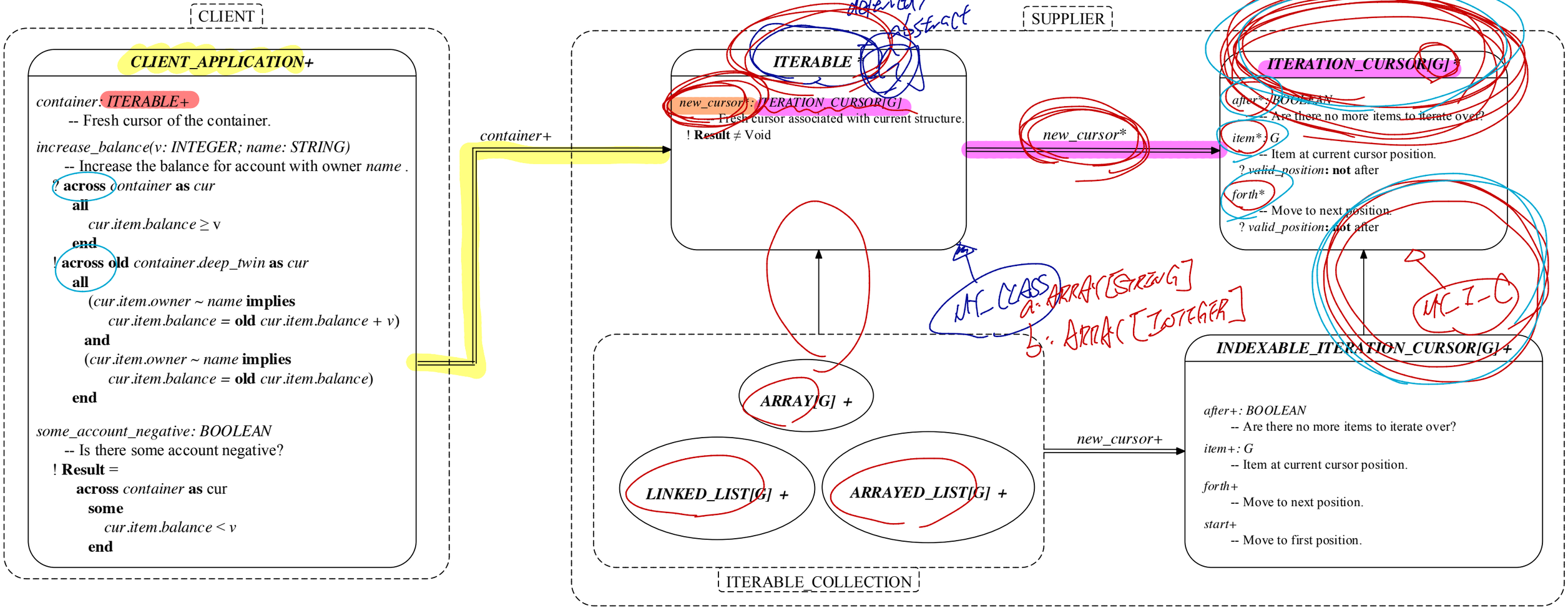
```
class
  CART
feature
  orders: ARRAY[ORDER]
end

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

## 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
```

Problems with the current design?


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
container+
SUPPLIER
ITERABLE +
new_cursor*: ITERATION_CURSOR[G]
-- Fresh cursor associated with current structure.
! Result ≠ Void
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
ARRAY[G] +
LINKED_LIST[G] +
ARRAYED_LIST[G] +
ITERABLE_COLLECTION
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.

# Implementing the ITERATOR Pattern: Easy Case

```
class
    CART
inherit  ITERABLE [ORDER]
feature {NONE} -- Information Hiding
    orders: ARRAY [ORDER]

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

# Implementing the ITERATOR Pattern: Hard Case

```
class
    BOOK[G]
inherit ITERABLE[ _____ ]
feature {NONE} -- Information Hiding
    names: ARRAY[STRING]
    records: ARRAY[G]

feature
    new_cursor: _____[_____]
        do

        end

end
```

TUPLE[STRING, G]