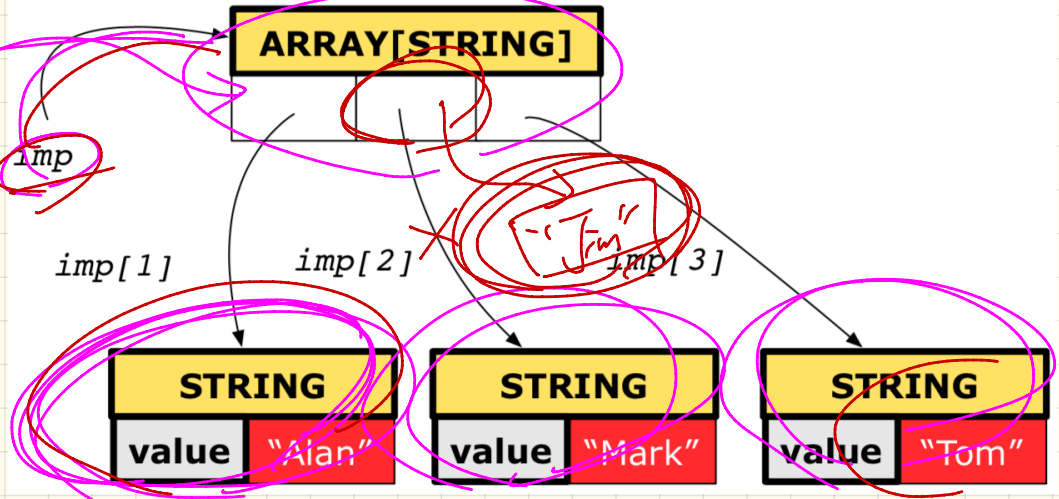
Monday January 21

Lecture 5

Copying Collection Objects : Reference Copy & Make Changes

```
1  old_imp := imp
2  Result := old_imp = imp   -- Result = true
3  imp[2] := "Jim"
4  Result :=
5    across 1 |..| imp.count as j
6    all imp [j.item] ~ old_imp [j.item]
7    end -- Result = true
```



Copying Collection Objects: Shallow Copy & Make 1st-level changes

```
1  old_imp := imp.twin
2  Result := old_imp = imp   -- Result = false
3  imp[2] := "Jim"
4  Result :=
5     across 1 |..| imp.count as j
6     all imp [j.item] ~ old_imp [j.item]
7  end -- Result = false
```

old_imp[2] ?

After (3):

old_imp[2] = imp[2]

"Jim"

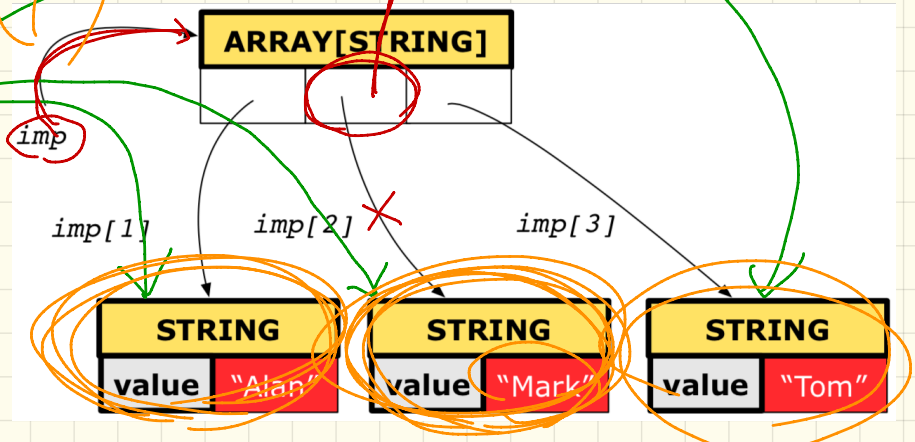
F

T
F
T

F

old_imp → ARRAY[STRING]

old_imp[1] := imp[1]



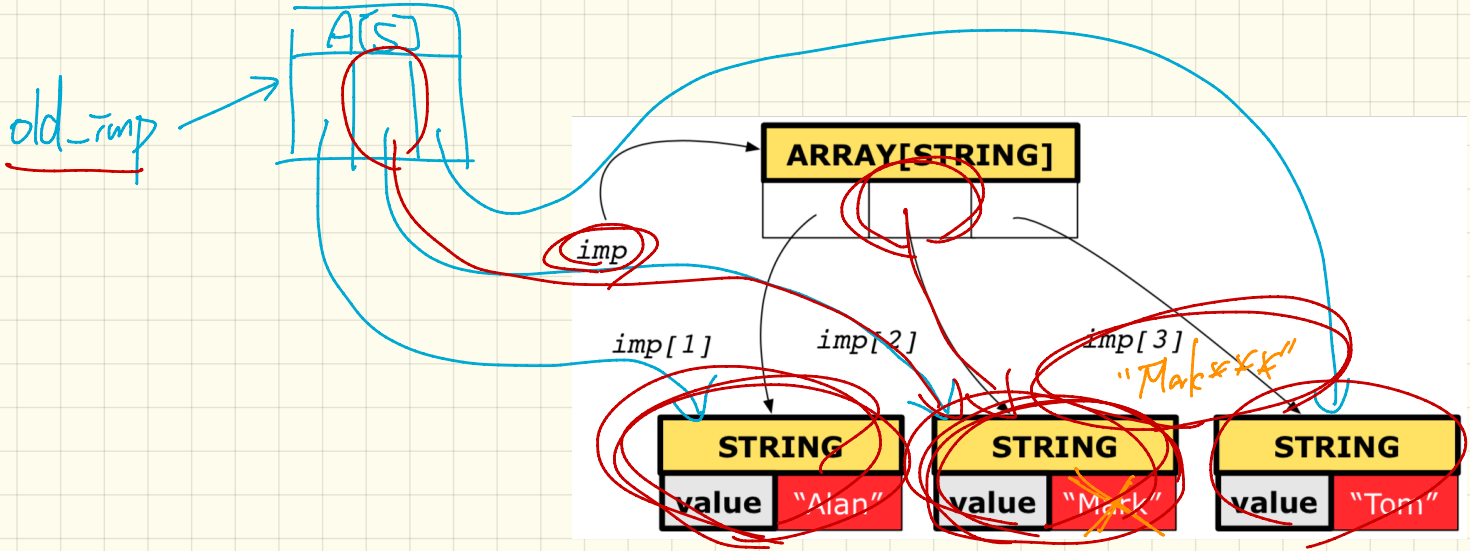
Copying Collection Objects: Shallow Copy & Make 2nd-level changes

```
1  old_imp := imp.twin
2  Result := old_imp ~ imp   -- Result = false
3  imp[2].append ("***")
4  Result :=
5    across 1 |..| imp.count as j
6    all imp [j.item] ~ old_imp [j.item]
7    end -- Result = true
```

After L3.

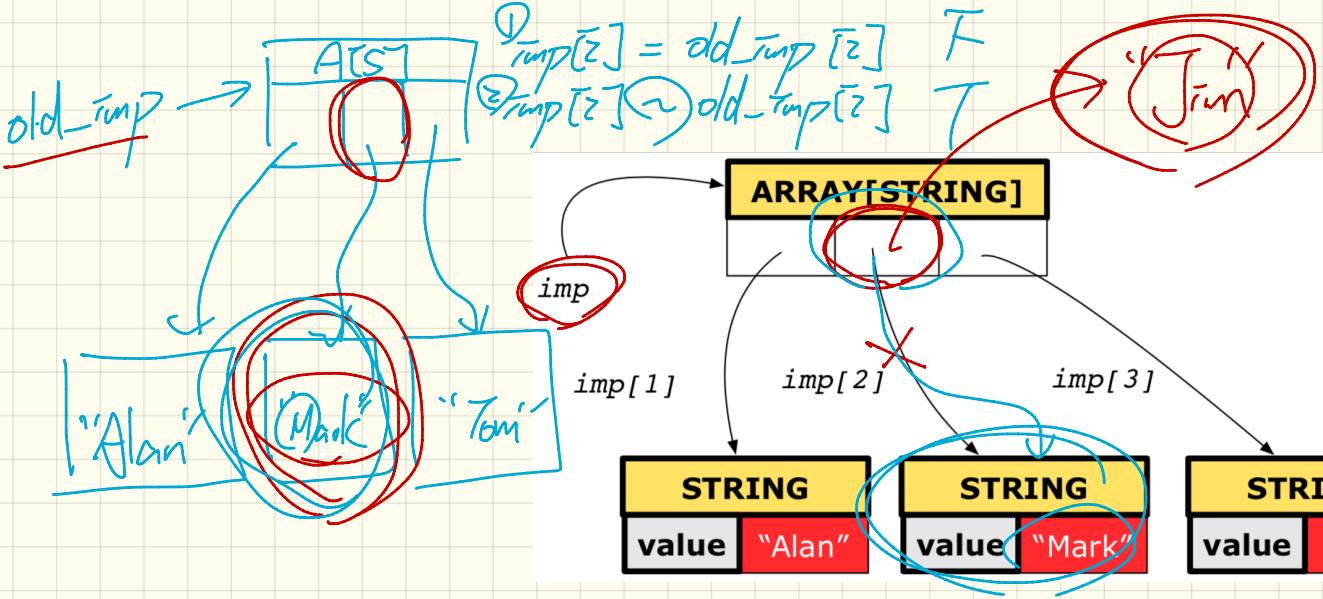
imp[2] = old_imp[2]

T



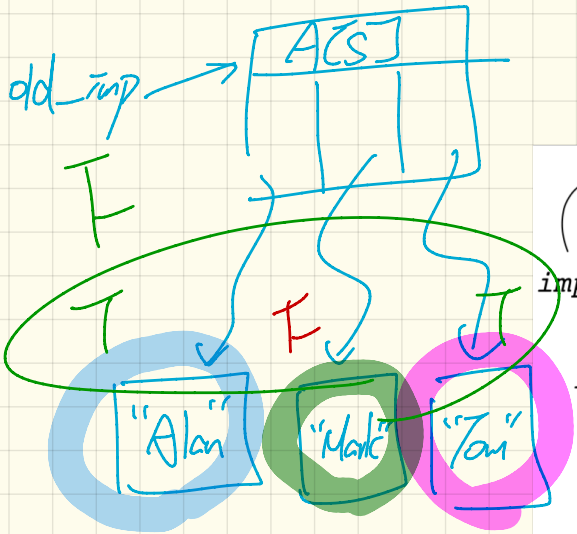
Copying Collection Objects: Deep Copy & Make 1st-level Changes

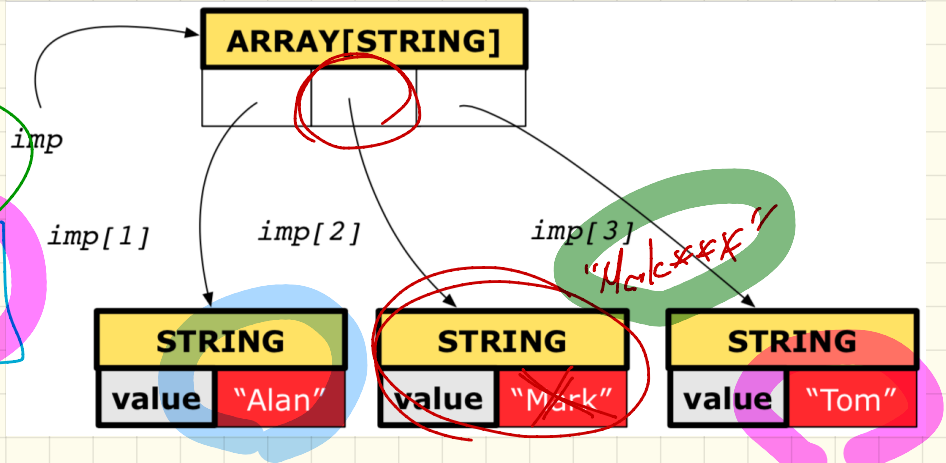
```
1  old_imp := imp.deep_twin
2  Result := old_imp = imp   -- Result = false
3  [illegible]
4  Result :=
5    across 1 |..| imp.count as j
6    all imp [j.item] ~ old_imp [j.item] end -- Result = false
```

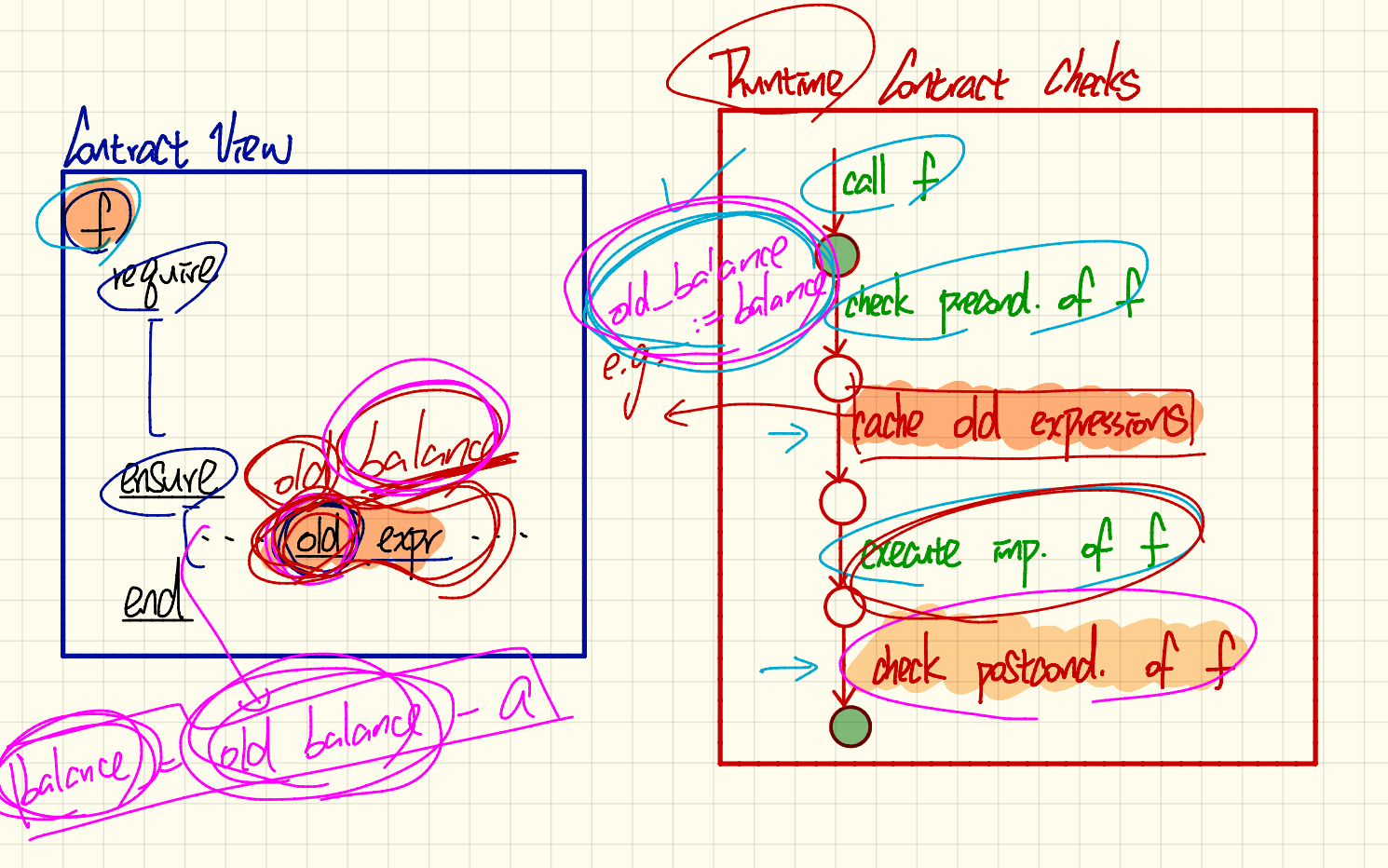


Copying Collection Objects: Deep Copy & Make 2nd-level changes

```
1  old_imp := imp.deep_twin
2  Result := old_imp = imp   -- Result = false
3  imp[2].append ("***")
4  Result :=
5    across 1 |..| imp.count as j
6    all imp [j.item] ~ old_imp [j.item] end -- Result = false
```

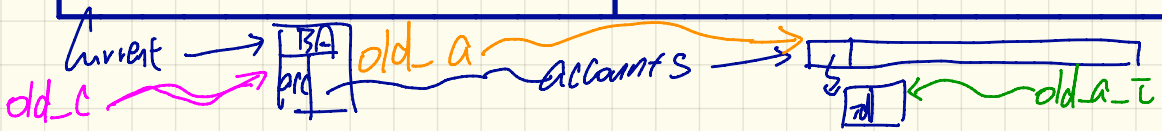





Contract View
f
require
ensure
old balance
(old) expr
end
balance = old balance - a
Runtime Contract Checks
call f
old_balance := balance
e.g.
check precond. of f
cache old expressions
execute imp. of f
check postcond. of f

Caching Values for old Expressions in Postconditions

ensure	How to Cache at Runtime?
old balance = balance - a	old_balance := balance
✓ old accounts[i].id	old_accounts_i_id := accounts[i].id
✓ (old accounts[i]).id	old_a_i := accounts[i]
✓ (old accounts)[i].id	old_a := accounts
✓ (old Current)accounts[i].id	old_C := Current



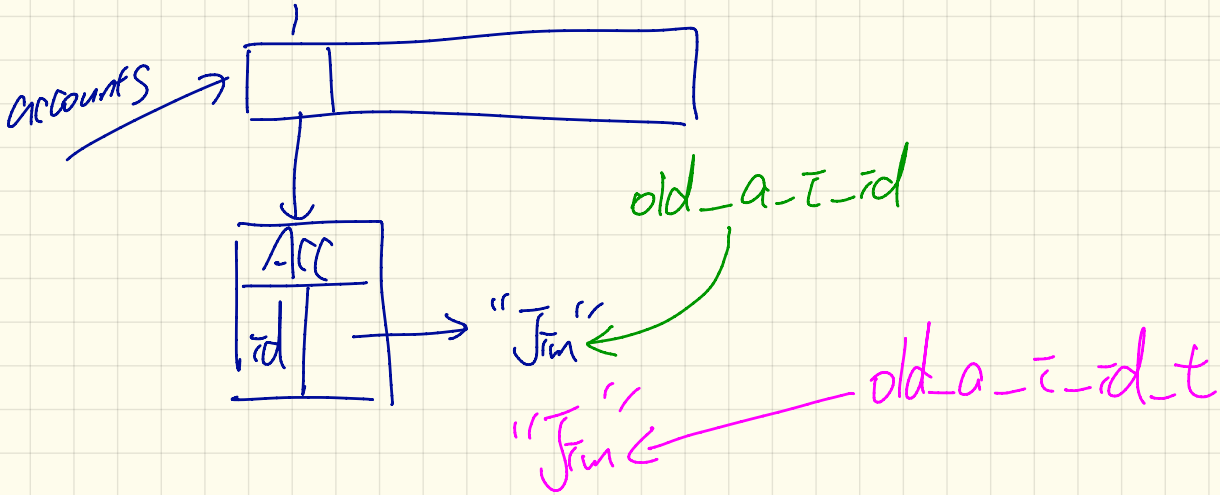
✓ old accounts[i].id

old_a_i_id := accounts[i].id

✓ old accounts[i].id.twin

old_a_i_id_t := accounts[i].id.twin

✓ old accounts[i].id.deep_twin



$$\forall s \mid s \in EECS331 \cdot s.pass$$

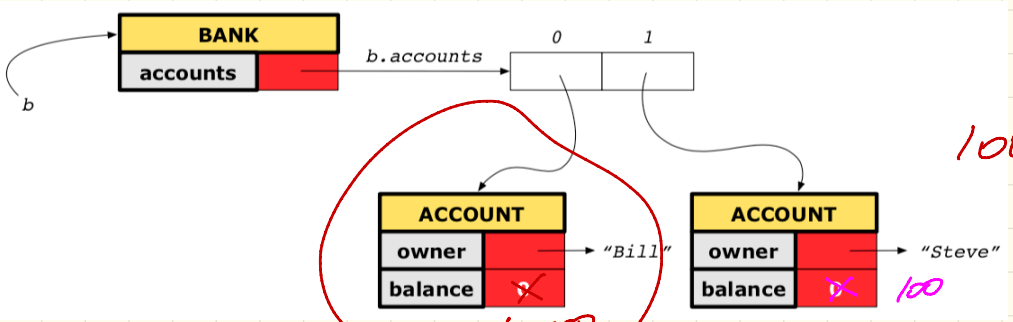
$$\equiv \neg(\exists s \mid s \in EECS331 \cdot \neg s.pass)$$

across accounts as acc
some
acc.item.ower ~ n
end

not (across accounts as acc
all
not (acc.item.ower ~ n)
end
)

Version 1: Incomplete Contracts, Correct Implementation

b.deposit ("Steve", 100)



100 = 0 + 100

(T)

100000

X

not caught.

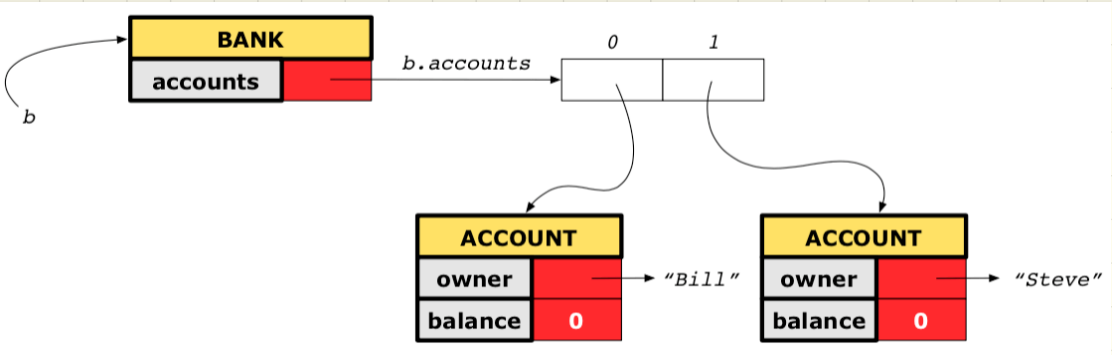
```
class BANK
  deposit_on_v1 (n: STRING; a: INTEGER)
    require across accounts as acc some acc.item.owner ~ n end
    local i: INTEGER
    do
      from i := accounts.lower
      until i > accounts.upper
      loop
        if accounts[i].owner ~ n then accounts[i].deposit(a) end
        i := i +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

to be cached in pre-state.

"Steve"

Version 2 : Incomplete Contracts, Wrong Implementation

b.deposit ("Steve", 100)

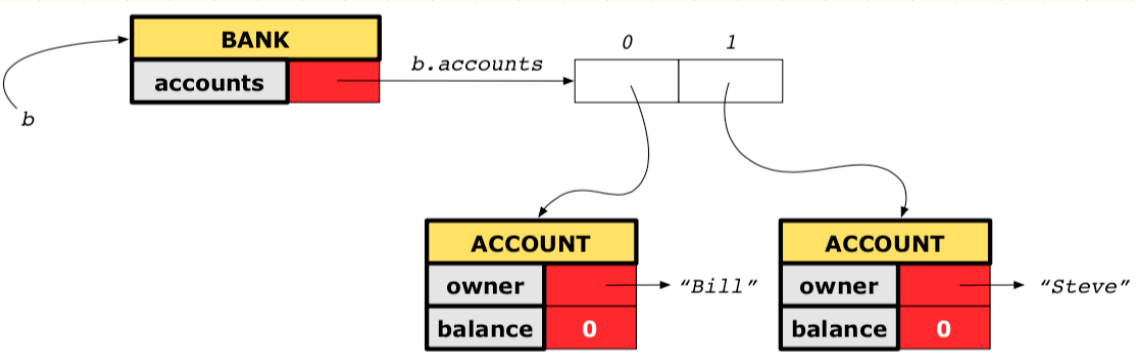



```
class BANK
  deposit_on_v2 (n: STRING; a: INTEGER)
    require across accounts as acc some acc.item.owner ~ n end
    local i: INTEGER
    do
      -- same loop as in version 1

      -- 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)




```
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
```