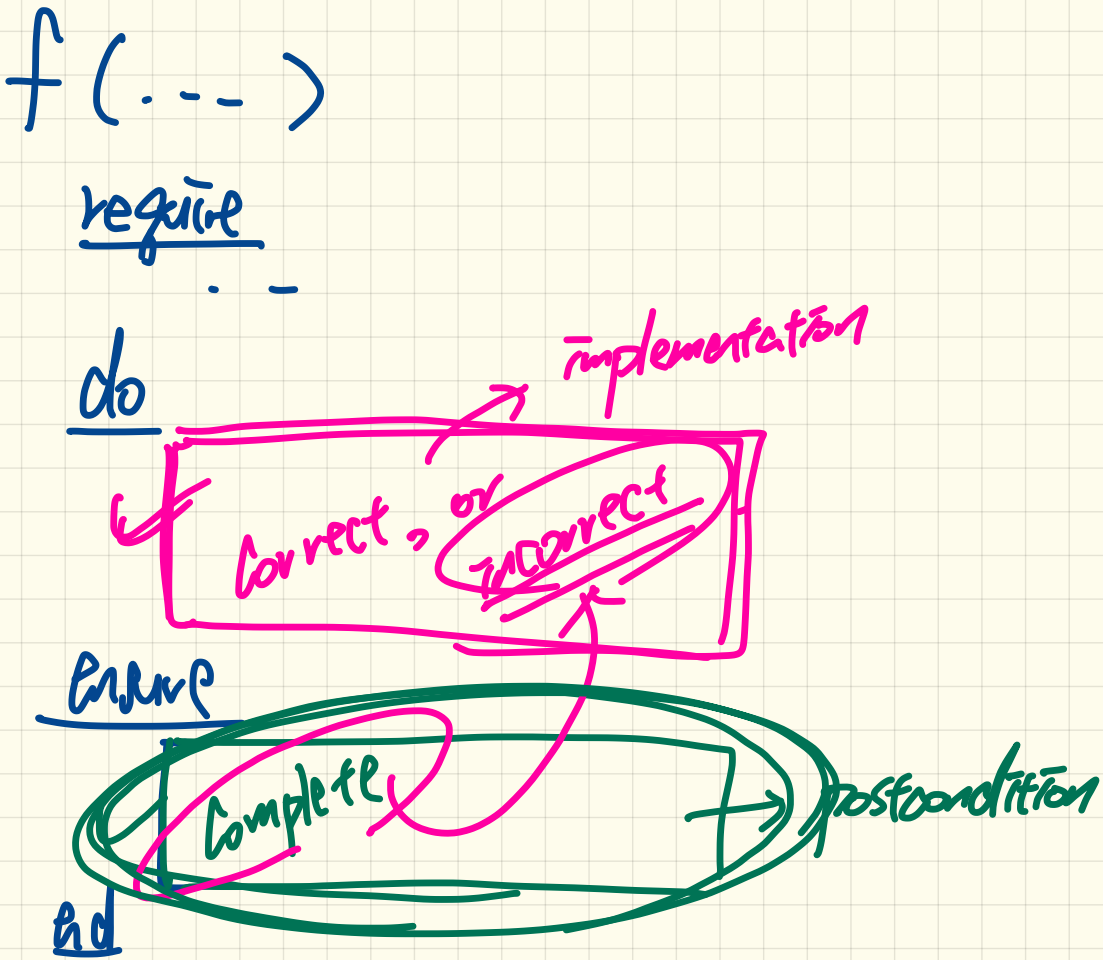
Lecture 7
Monday January 27


f(...)
require
...
do
implementation
correct, or
incorrect
ensure
complete
postcondition
end

```
class BANK
create make
feature
  accounts: ARRAY[ACCOUNT]
  make do create accounts.make_empty end
  account_of (n: STRING): ACCOUNT
    require -- the input name exists
      existing: across accounts is acc some acc.owner ~ n end
        -- not (across accounts is acc all acc.owner /~ n end)
    do ... ensure Result.owner ~ n end
  add (n: STRING)
    require -- the input name does not exist
      non_existing: across accounts is acc all acc.owner /~ n end
        -- not (across accounts is acc some acc.owner ~ n end)
    local new_account: ACCOUNT
    do
      create new_account.make (n)
      accounts.force (new_account, accounts.upper + 1)
    end
end
```

```
class
  ACCOUNT

inherit
  ANY
    redefine is_equal end

create
  make

feature -- Attributes
  owner: STRING
  balance: INTEGER

feature -- Commands
  make (n: STRING)
    do
      owner := n
      balance := 0
    end

  deposit(a: INTEGER)
    do
      balance := balance + a
    ensure
      balance = old balance + a
    end

  is_equal(other: ACCOUNT): BOOLEAN
    do
      Result :=
            owner ~ other.owner
        and balance = other.balance
    end
end
```

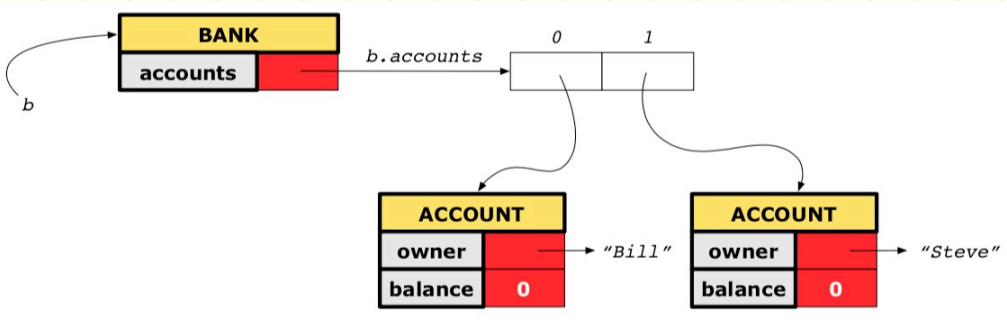
Unit Test for All 5 Versions

```
class TEST_BANK
  test_bank_deposit_correct_imp_incomplete_contract: BOOLEAN
    local
      b: BANK
    do
      comment("t1: correct imp and incomplete contract")
      create b.make
      b.add ("Bill")
      b.add ("Steve")

      -- deposit 100 dollars to Steve's account
      b.deposit_on_v1 ("Steve", 100)
      Result :=
           b.account_of("Bill").balance = 0
        and b.account_of("Steve").balance = 100
      check Result end
  end
end
```

Version 1: Incomplete Contracts, Correct Implementation

b.deposit("Steve", 100)

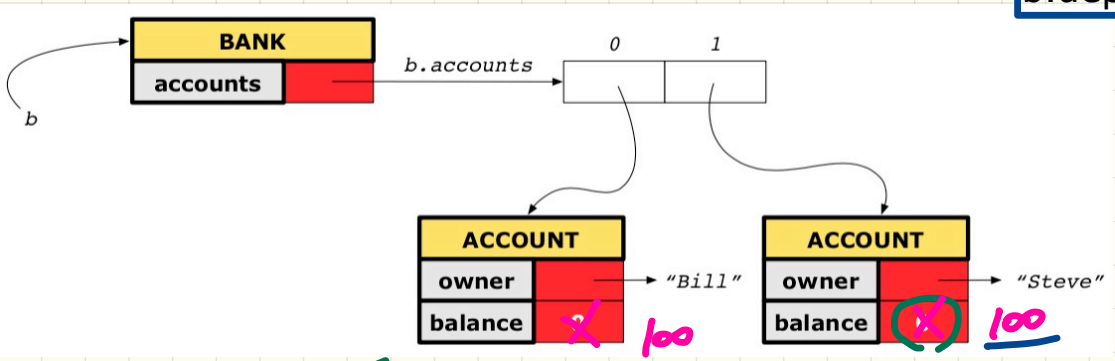



```
class BANK
  deposit_on_v1 (n: STRING; a: INTEGER)
    require across accounts is acc some acc.owner ~ n end
    local i: INTEGER
    do
      from i := accounts.lower
      until i > accounts.upper
      loop
        if accounts[i].owner ~ n then accounts[i].deposit(a) end
        i := i + 1
  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

Correct

Version 2: Incomplete Contracts, Wrong Implementation

b.deposit("Steve", 100)




```
class BANK
  deposit_on_v2 (n: STRING; a: INTEGER)
    require across accounts is acc some acc.owner ~ n end
    local i: INTEGER
    do
      ...
      -- imp. of version 1, followed by a deposit into 1st account
      accounts[accounts.lower].deposit(a)
    ensure
      num_of_accounts_unchanged:
        accounts.count = old accounts.count
      balance_of_n_increased:
        Current.account_of(n).balance =
          old Current.account_of(n).balance + a
    end
end
```

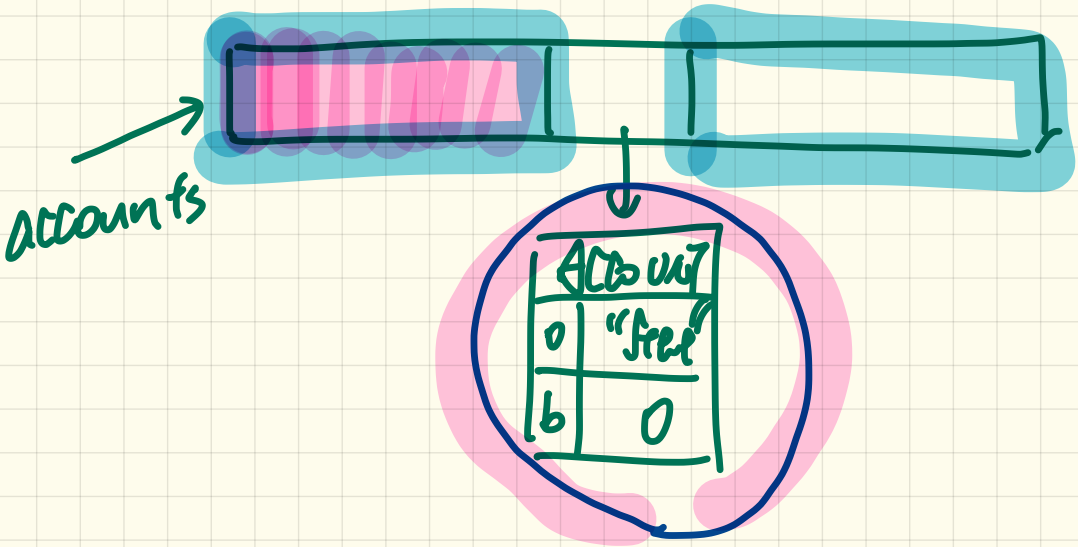
old_value := 0

Steve 100

100 = 0 + 100

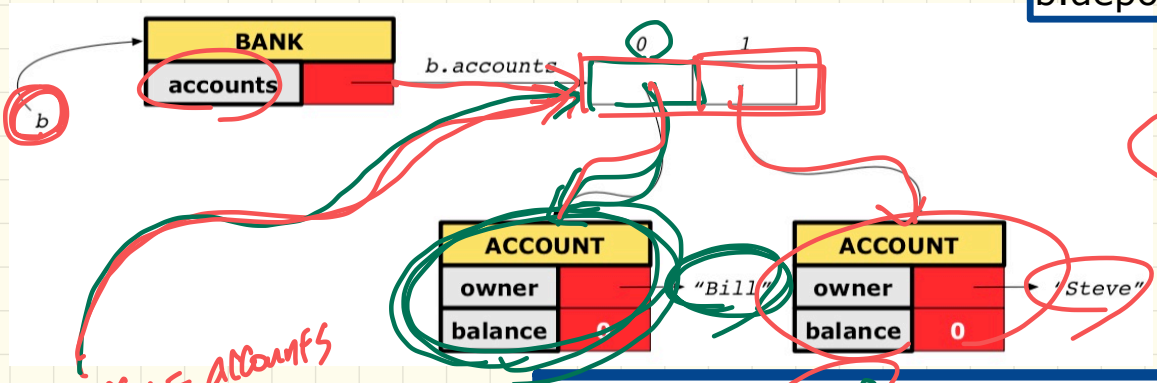
T

cache int value


accounts
Account
0
"free"
6
0

Version 3: Complete Contracts (Ref. Copy), Correct Implementation

b.deposit("Steve", 100)



old_accs := accounts

```
class BANK
  deposit_on_v3 (n: STRING; a: INTEGER)
    require across accounts is acc some acc.owner ~ n end
    local i: INTEGER
    do ...
      -- imp. of version 1, followed by a deposit into 1st account
      accounts[accounts.lower].deposit(a)
    ensure
      num_of_accounts_unchanged: accounts.count = old accounts.count
      balance_of_n_increased:
        Current.account_of(n).balance =
          old Current.account_of(n).balance + a
      others_unchanged:
        across old accounts is acc
        all
          acc.owner /~ n implies acc ~ Current.account_of(acc.owner)
        end
    end
end
```

1st iter.

Bill /~ Steve ⇒ T

2nd iter.

Steve /~ Steve ⇒ F

T

T ⇒ _ ≡ T

"old" version of account

"new" version of acc.

acc's owner is not the one to be changed

Use of across in Postcondition

Version 1

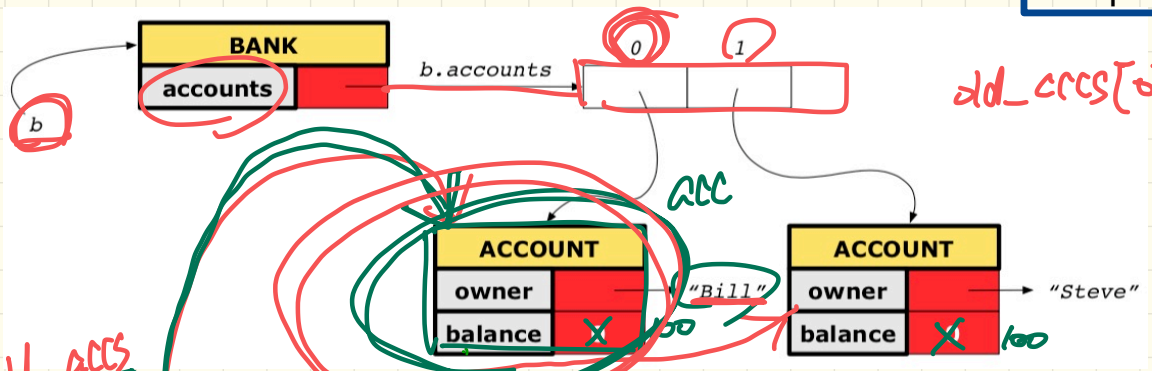
```
across old accounts is acc
all
   acc.owner /~ n
   implies
   acc ~ Current.account_of (acc.owner)
end
```

Version 2

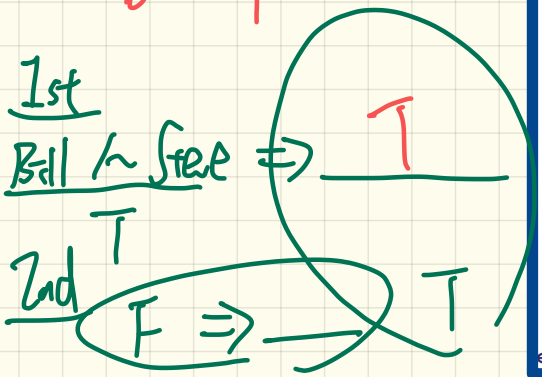
```
across (old accounts.lower |..| old accounts.upper) is i
all
   (old accounts)[i].owner /~ n
   implies
   (old accounts)[i] ~ Current.account_of ( (old accounts)[i].owner )
end
```

Version 4: Complete Contracts (Shallow Copy), Correct Implementation

b.deposit("Steve", 100)

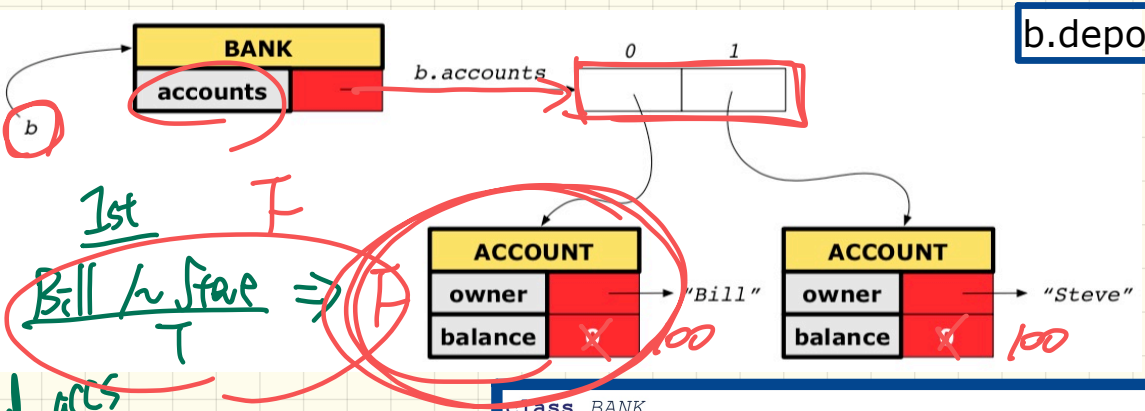



```
class BANK
  deposit_on_v4 (n: STRING; a: INTEGER)
    require across accounts is acc some acc.owner ~ n end
    local i: INTEGER
    do ...
      -- imp. of version 1, followed by a deposit into 1st account
      accounts[accounts.lower].deposit(a)
    ensure
      num_of_accounts_unchanged: accounts.count = old accounts.count
      balance_of_n_increased:
        Current.account_of(n).balance =
          old Current.account_of(n).balance + a
      others_unchanged:
        across old accounts.twin is acc
        all
          acc.owner /~ n implies acc ~ Current.account_of(acc.owner)
        end
    end
end
```

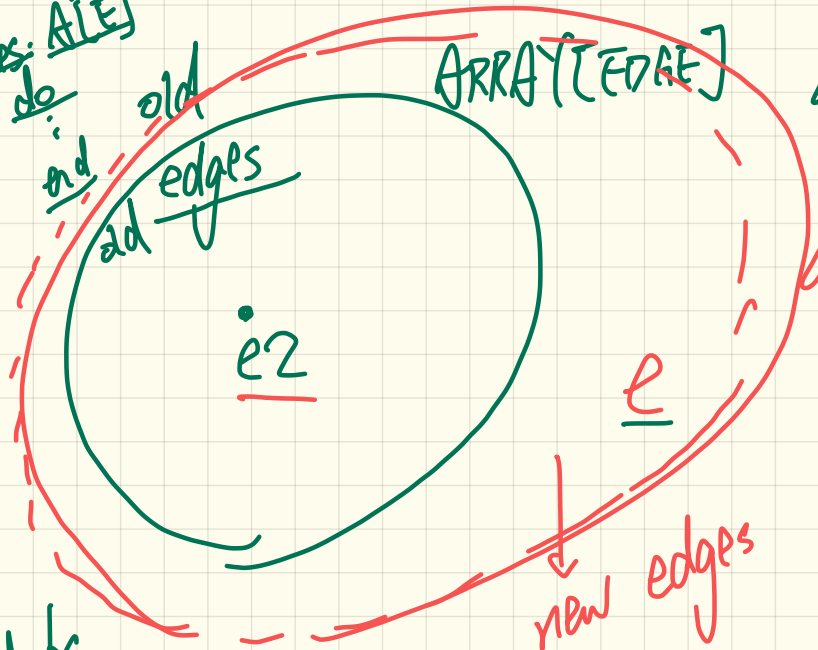


Version 5: Complete Contracts (Deep Copy), Correct Implementation

b.deposit("Steve", 100)




```
class BANK
  deposit_on_v5 (n: STRING; a: INTEGER)
    require across accounts is acc some acc.owner ~ n end
    local i: INTEGER
    do ...
      -- imp. of version 1, followed by a deposit into 1st account
      accounts[accounts.lower].deposit(a)
    ensure
      num_of_accounts_unchanged: accounts.count = old accounts.count
      balance_of_n_increased:
        Current.account_of(n).balance =
          old Current.account_of(n).balance + a
      others_unchanged :
        across old accounts.deep_twin is acc
        all
          acc.owner /~ n implies acc ~ Current.account_of(acc.owner)
        end
    end
end
```



add_edge (e)

add_edge (e)

ensure

count_incremented:

edges.count =

old edges.count

+ 1

edges

do

end

old edges ⊂ new edges

changed : _____

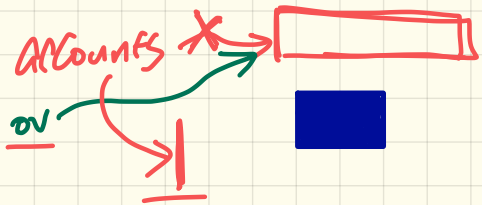
unchanged :

across old edges as e2

all e → current.has_edge(e2)

old edges.do_t

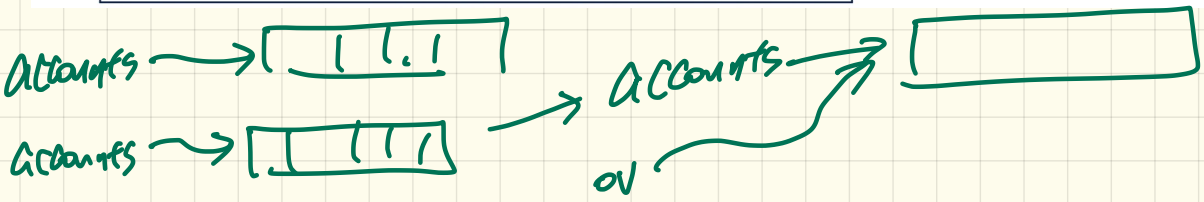
Complete Postcondition: Exercise



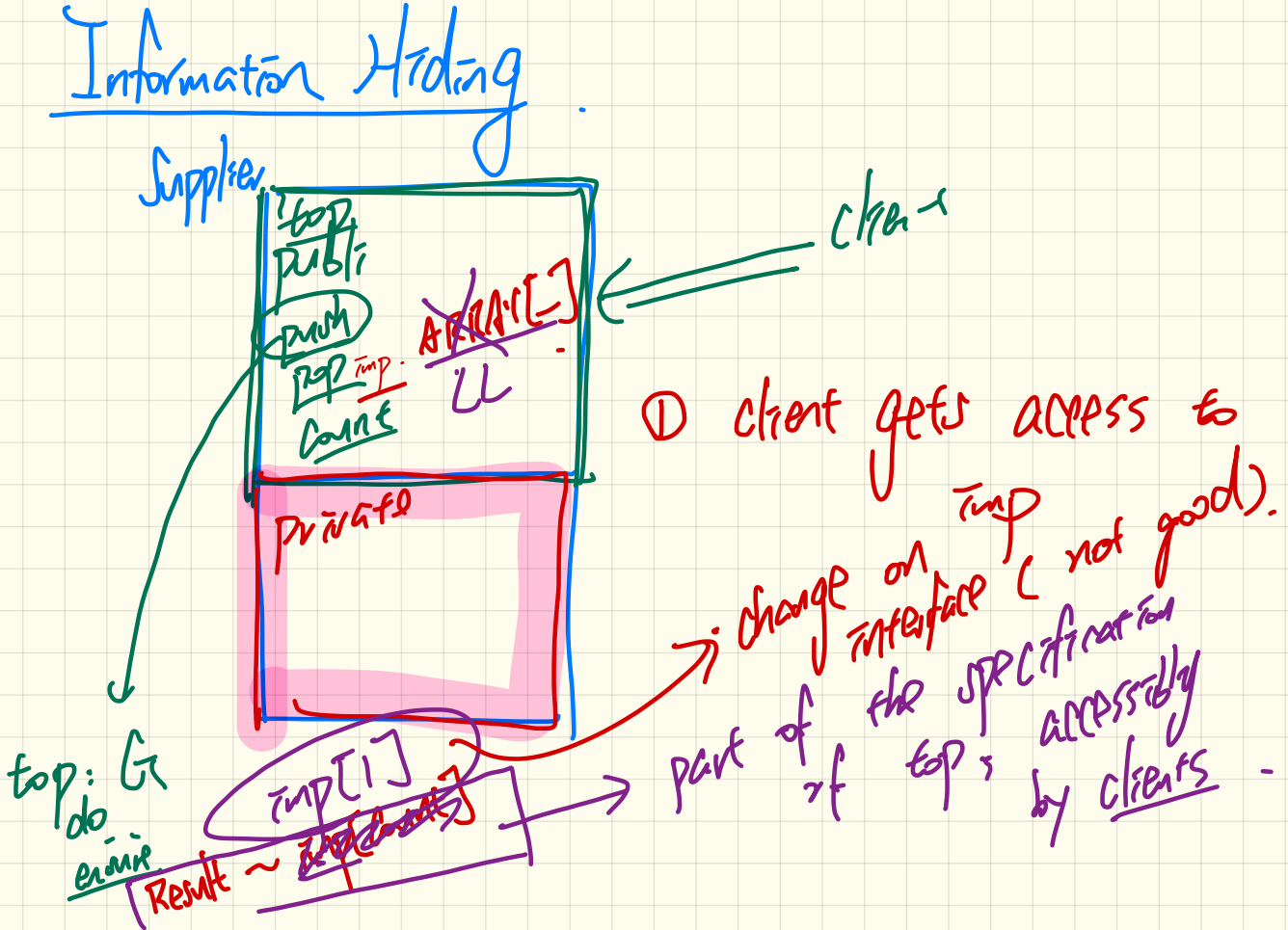
Consider the query *account_of (n: STRING)* of *BANK*.

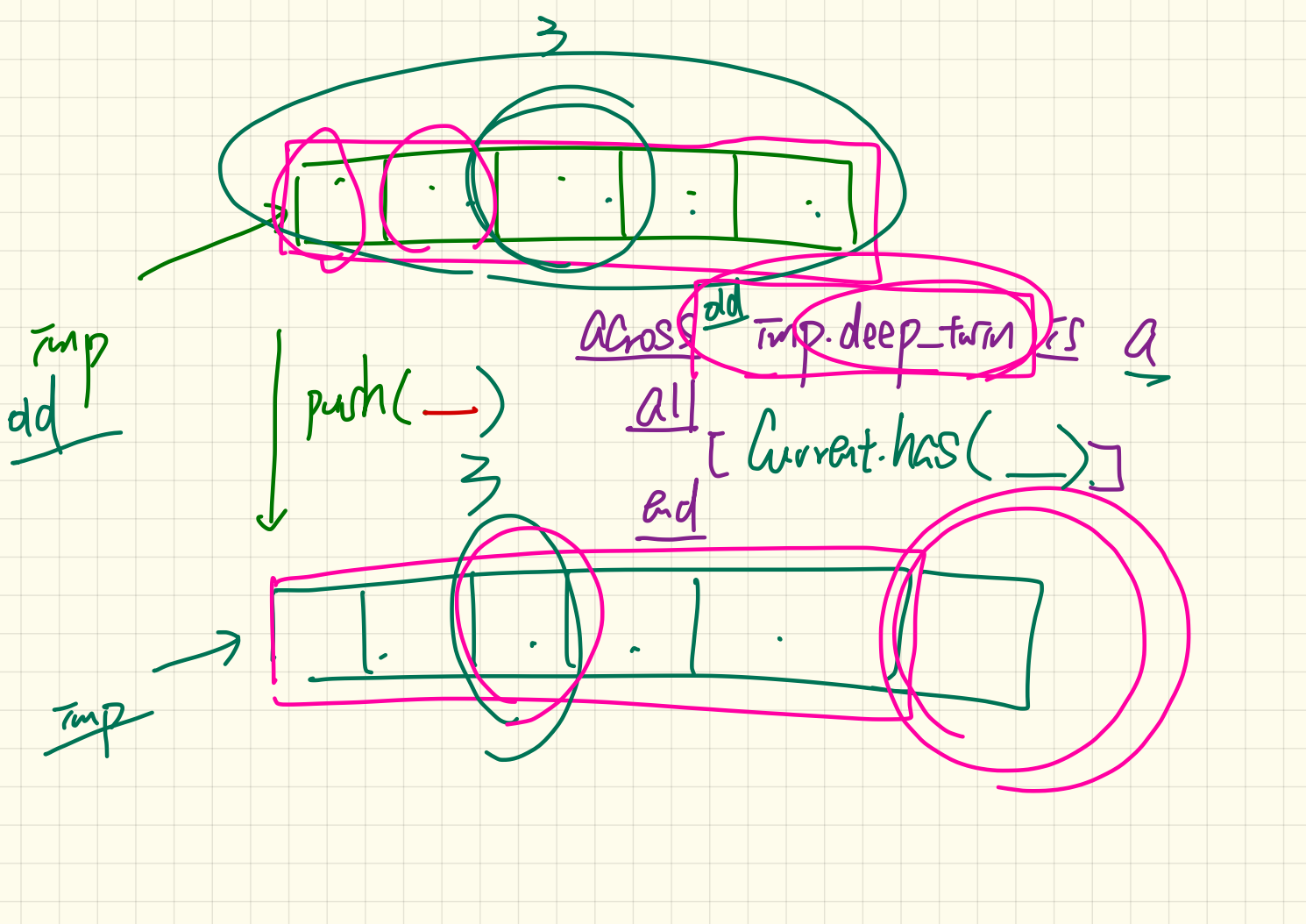
How do we specify (part of) its postcondition to assert that the state of the bank remains unchanged:

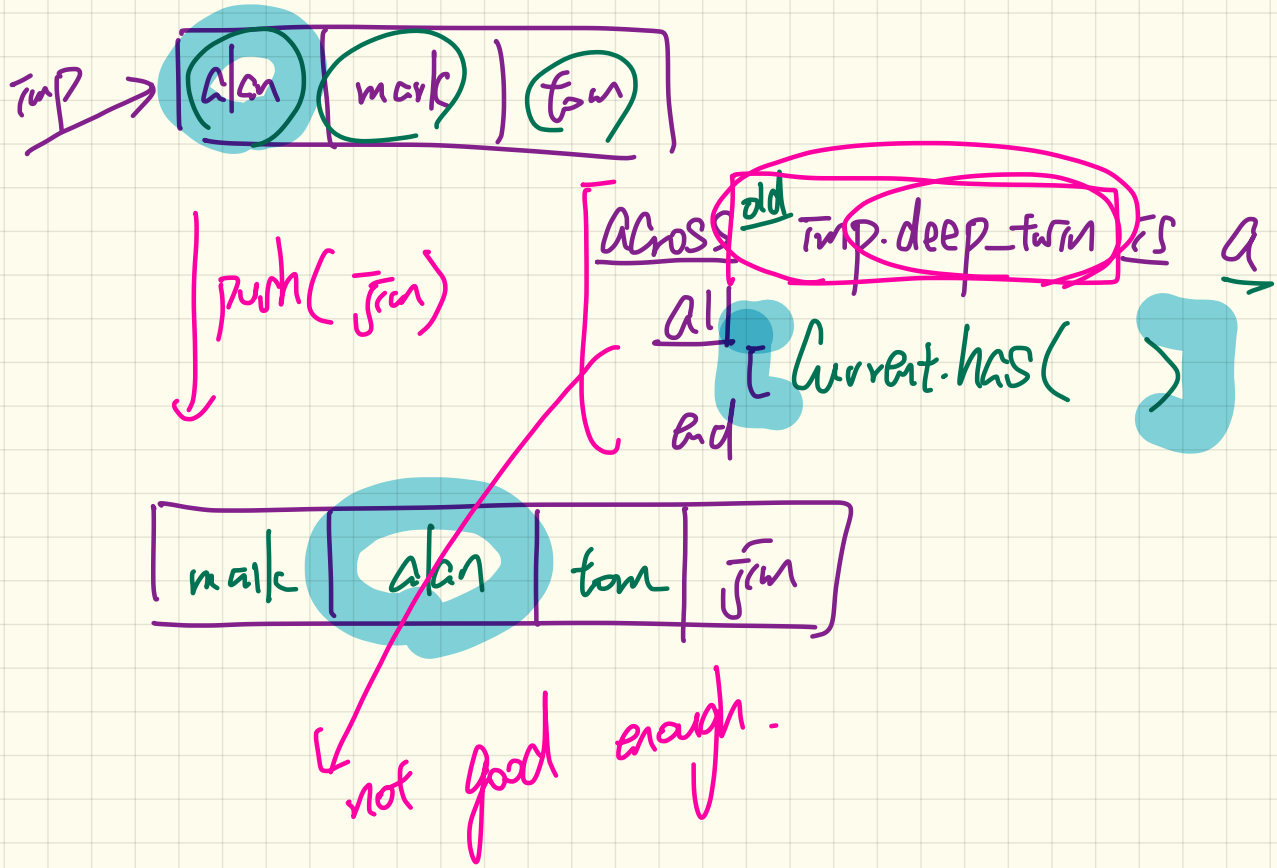
- `accounts = old accounts`
- `accounts = old accounts.twin`
- `accounts = old accounts.deep_twin`
- `accounts ~ old accounts`
- `accounts ~ old accounts.twin`
- `accounts ~ old accounts.deep_twin`

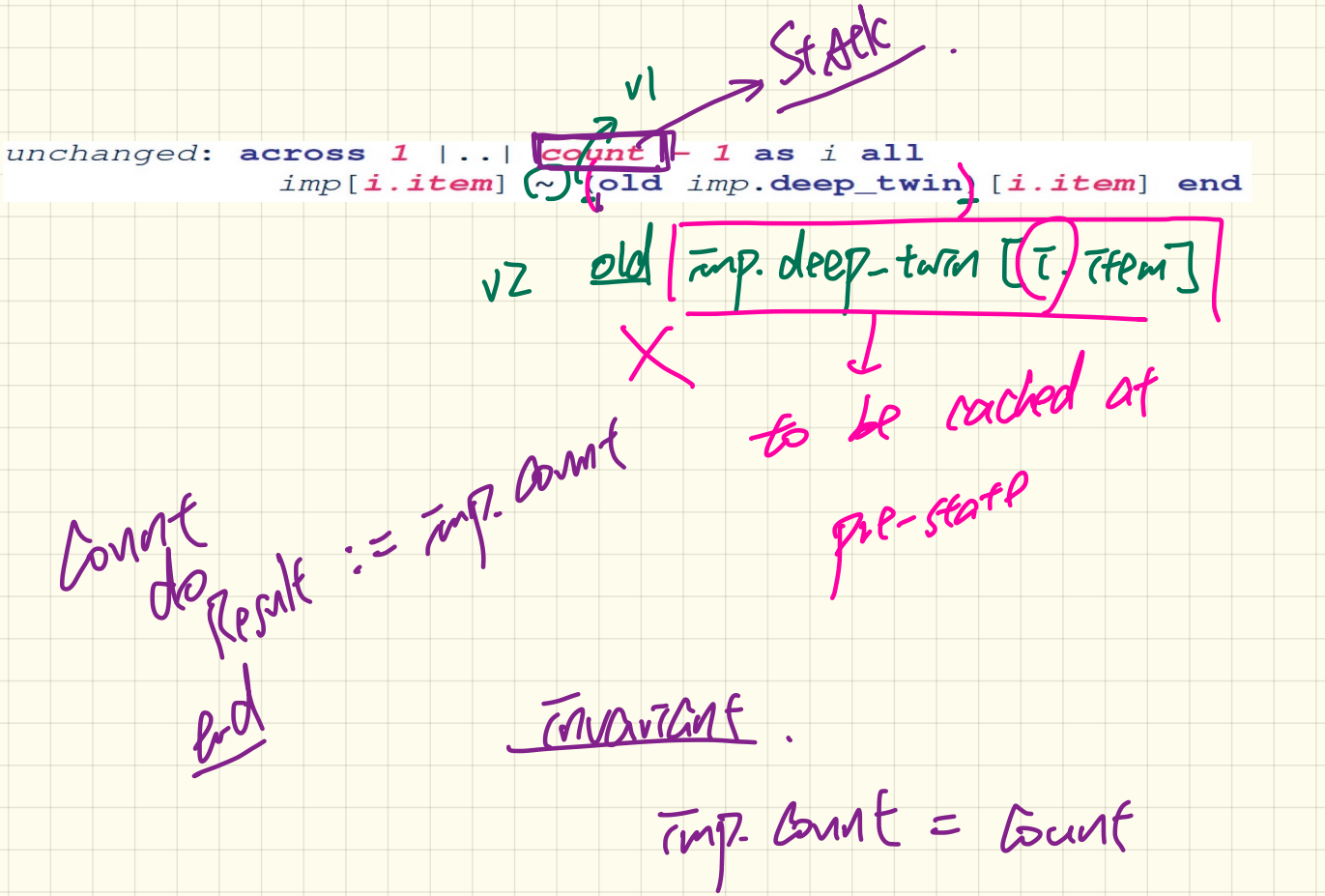


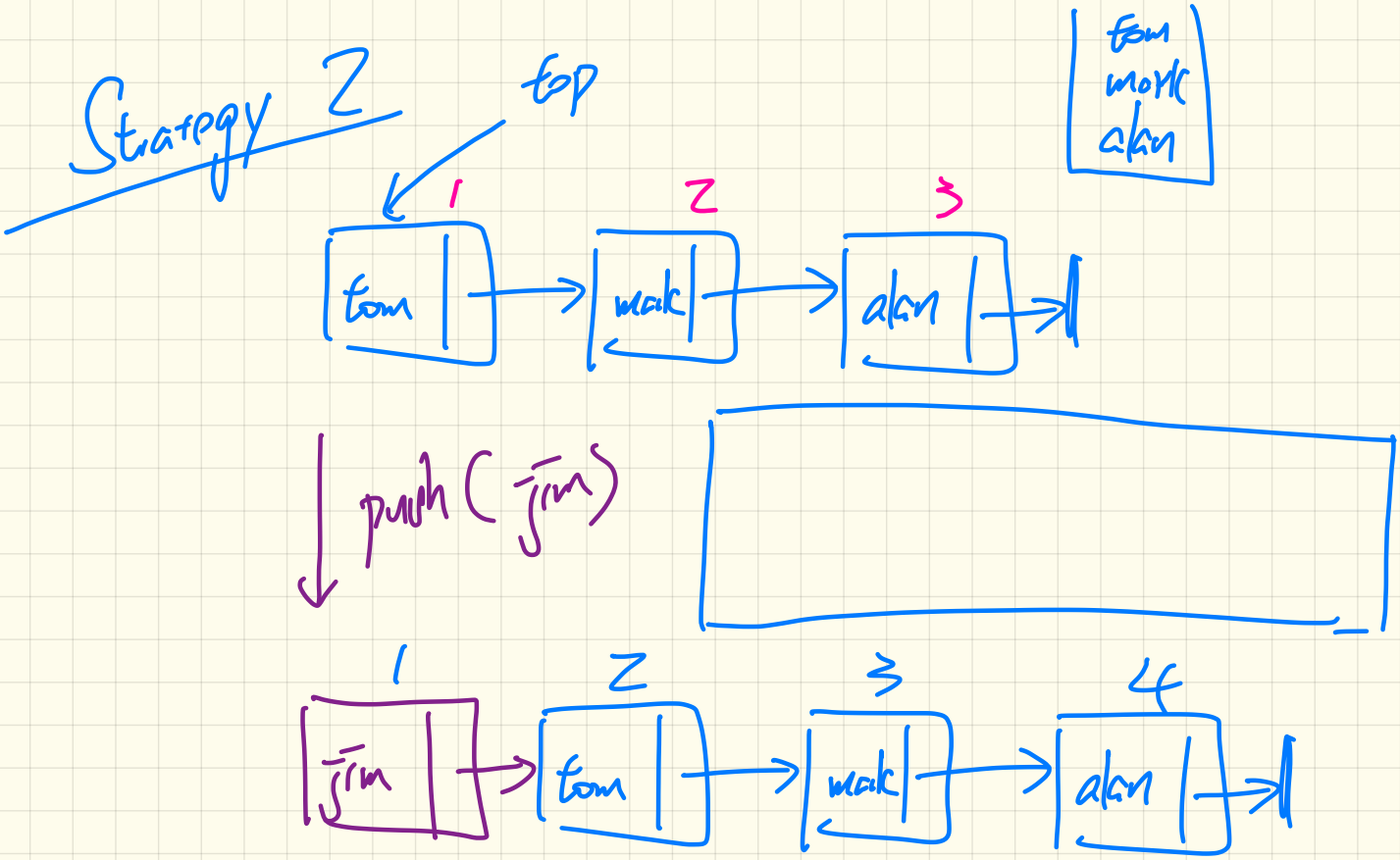
Information Hiding




tmp
add
push(_)
across
add
tmp.deep_twin
is
a
all
[current.has(_)]
end
tmp


tmp
alan
mark
tom
push(jim)
across
all
end
add
tmp.deep_twin
as
a
current.has(
mark
alan
tom
jim
not good enough.


unchanged: across 1 |..| count - 1 as i all
imp[i.item] ~ old imp.deep_twin [i.item] end
v1
Static
v2
old
imp.deep_twin [i.item]
to be cached at pre-state
Count do Result := imp.count end
invariant
imp.count = count


Strategy 2
top
1
2
3
tom
mark
alan
tom
mark
alan
push (jim)
1
2
3
4
jim
tom
mark
alan