Lecture 6

Wednesday January 22

Lab1: Due at 3pm this Friday

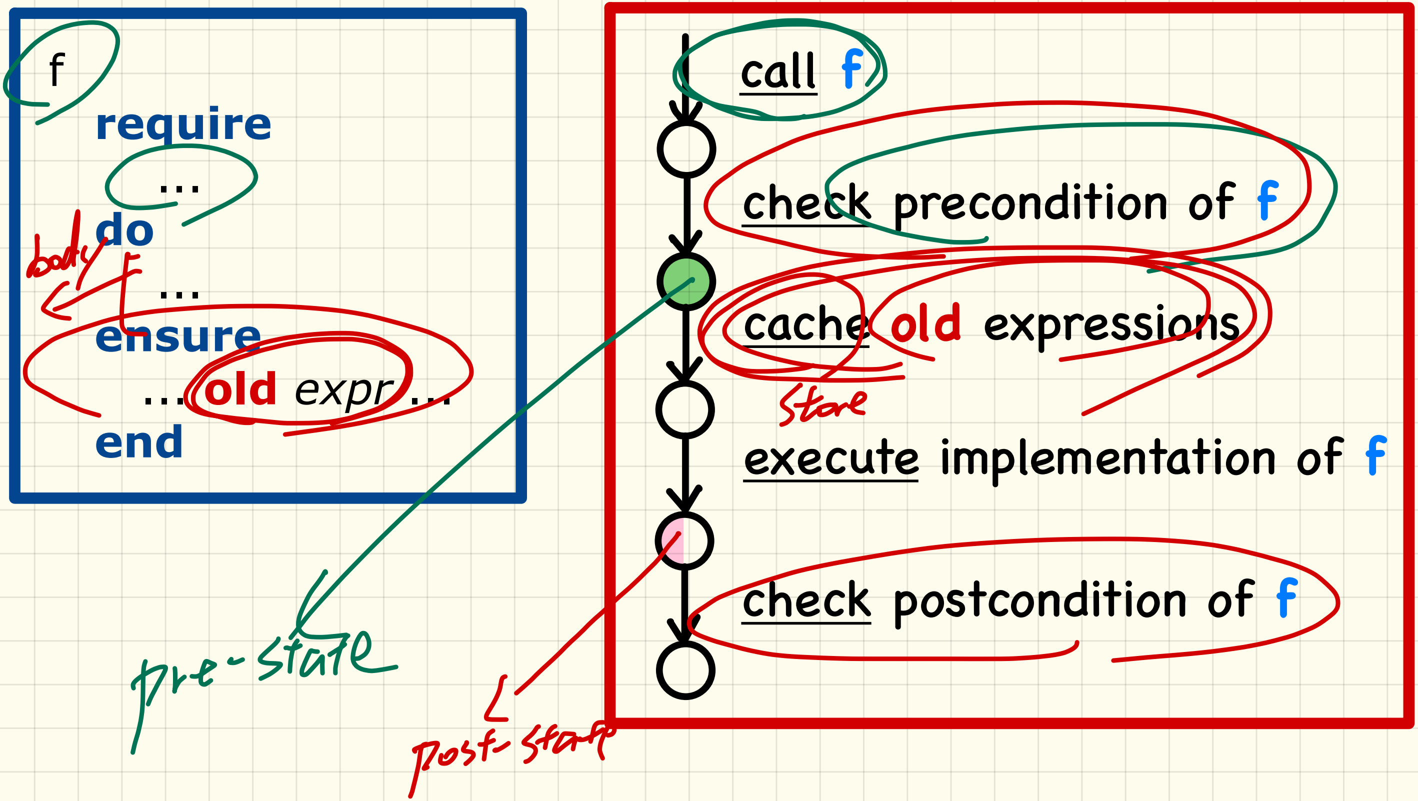
TA Office Hours: 12pm - 2pm LAS 2056

My office hours: 3pm to 5pm, Wednesday

Extra office hours: 3:30pm to 5:30pm, Thursday

Contract View

Runtime Contract Checks



```
f
    require
        ...
    do
        ...
    ensure
       ... old expr ...
    end
```

call f

check precondition of f

cache old expressions

execute implementation of f

check postcondition of f

Caching Values for old Expressions in Postconditions

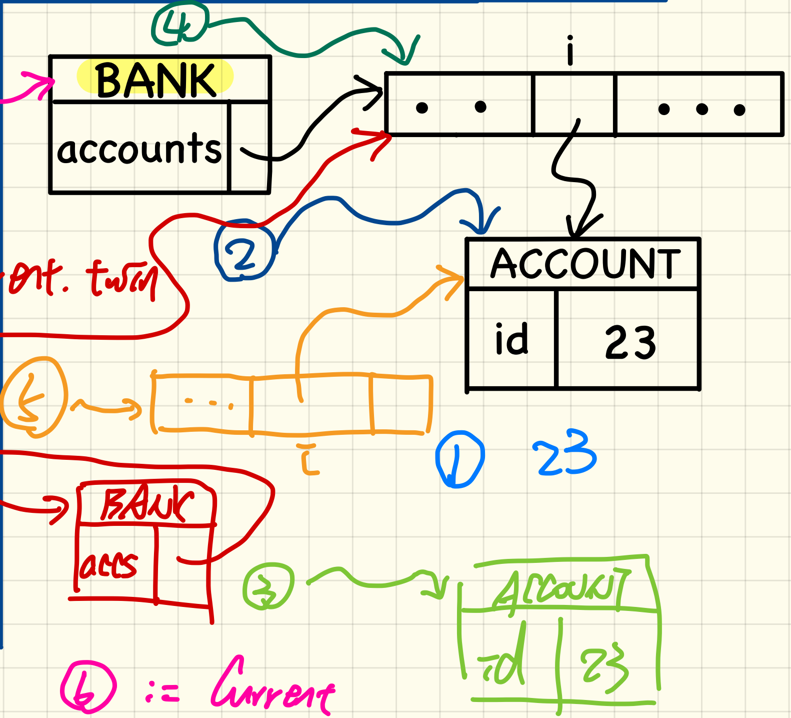
ensure (in context of BANK)

1. old accounts[i].id
2. (old accounts[i]).id
3. (old accounts[i].twin).id
4. (old accounts)[i].id
5. (old accounts.twin)[i].id
6. (old Current).accounts[i].id
7. (old Current.twin).accounts[i].id

this

⑦ := Current.twin

How to cache at runtime?



② := accounts[i]

③ := accounts[i].twin

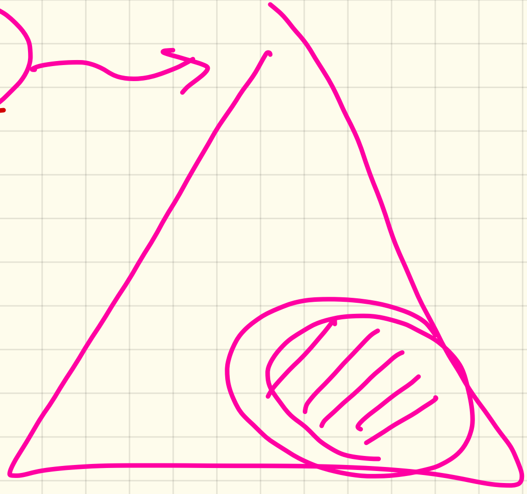
ACCOUNT

⑥ := Current

④ := accounts

⑤ := accounts.twin

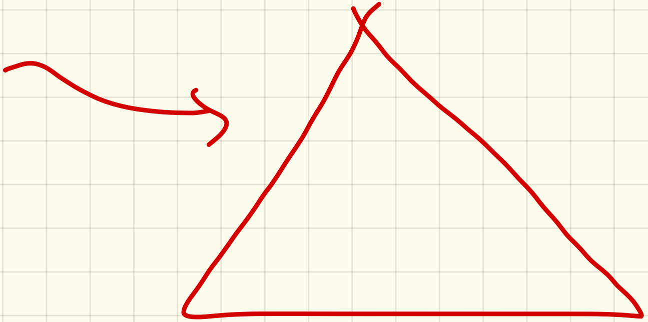
current



bst.insert(. .)

o_dt

h.

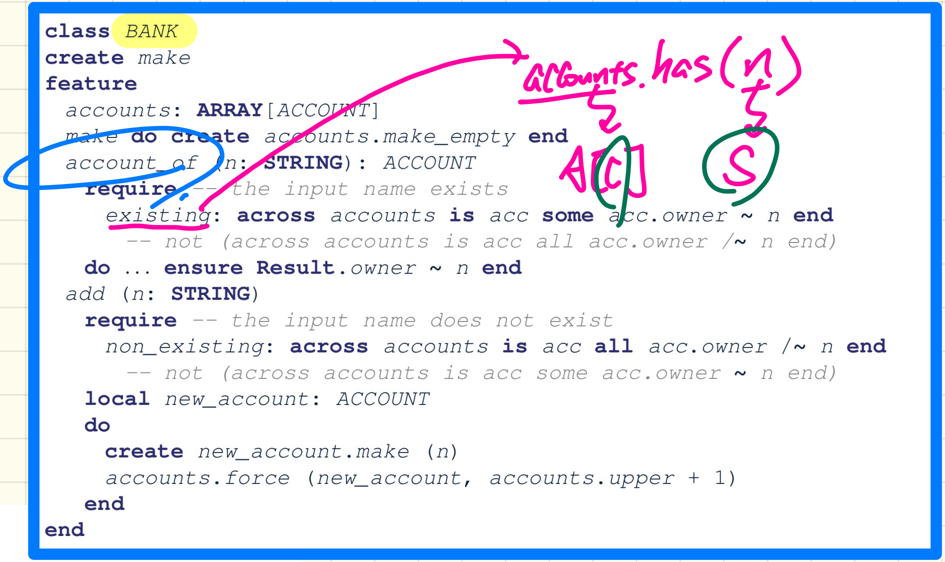


```
class BANK
    accounts: ARRAY[ACCOUNT]
end

class ACCOUNT
    id: INTEGER
end
```

accounts

accounts[i]

accounts[i].id

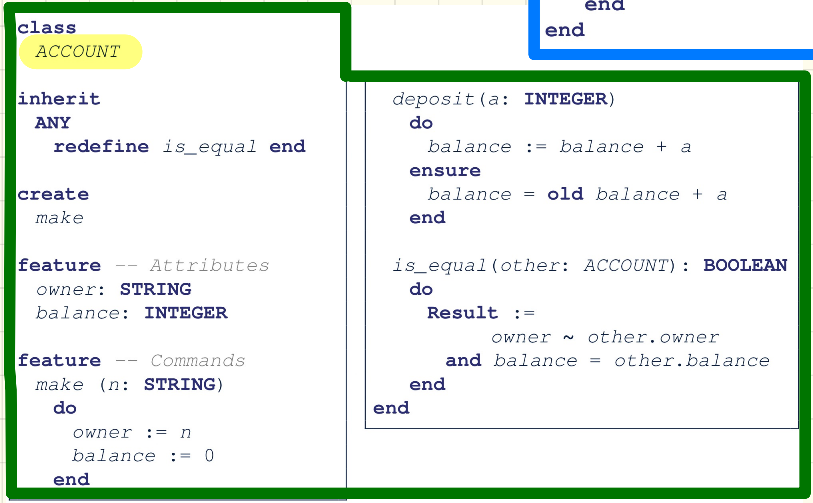


```
class BANK
create make
feature
  accounts: ARRAY[ACCOUNT]
  make do create accounts.make_empty end
  account_of (n: STRING): ACCOUNT
    require -- the input name exists
      existing: across accounts is acc some acc.owner ~ n end
        -- not (across accounts is acc all acc.owner /~ n end)
    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
```

accounts.has(n)

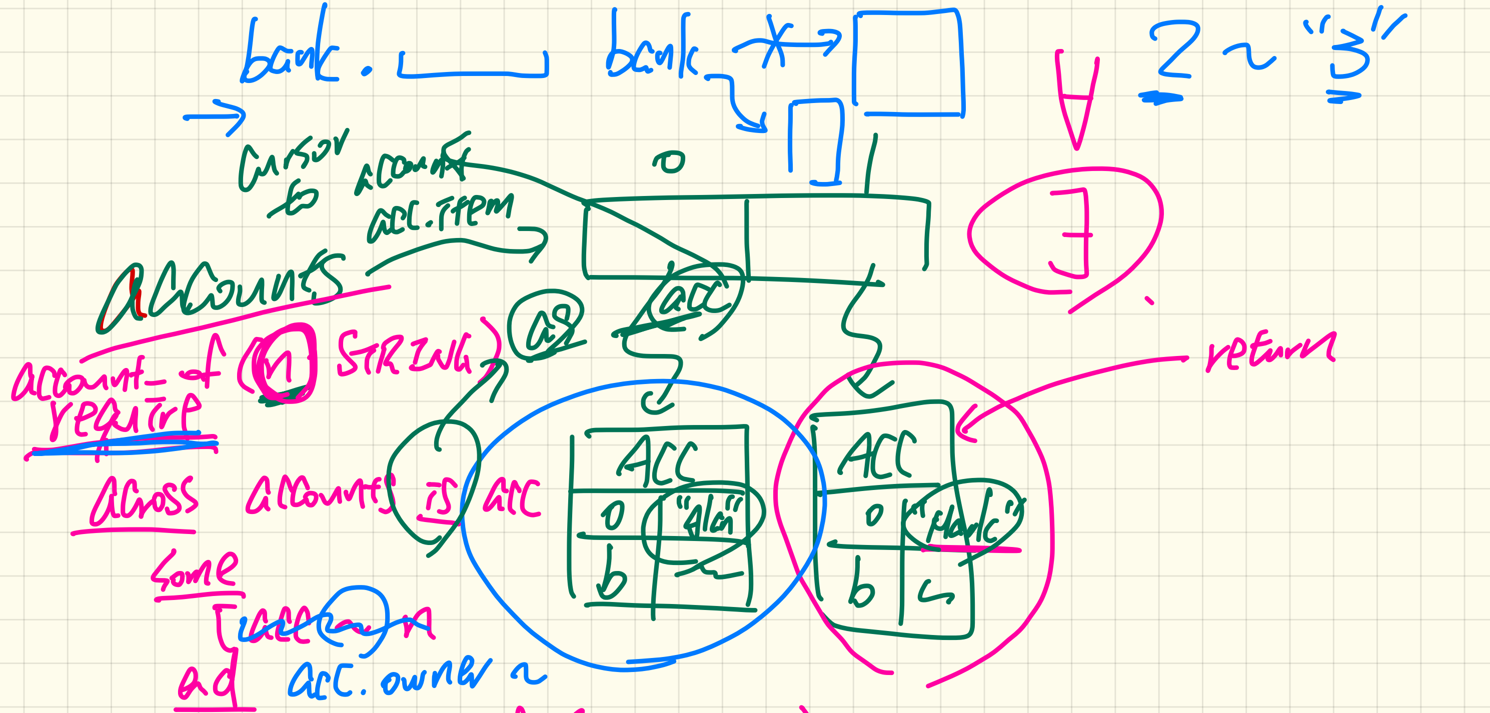
A[C]

S



```
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
```



→ cursor to account

accounts

account_of (n STRING)

require

across accounts is acc

some

~~[acc.n~~

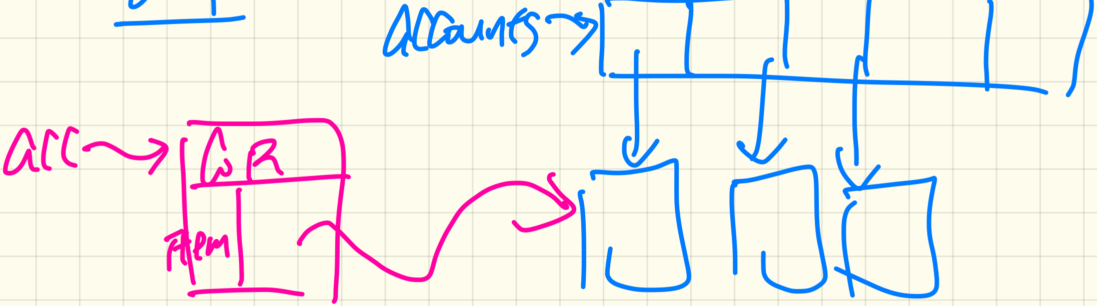
ad acc.owner ~

bank.account_of("Mark")

bank.account_of("Tom")

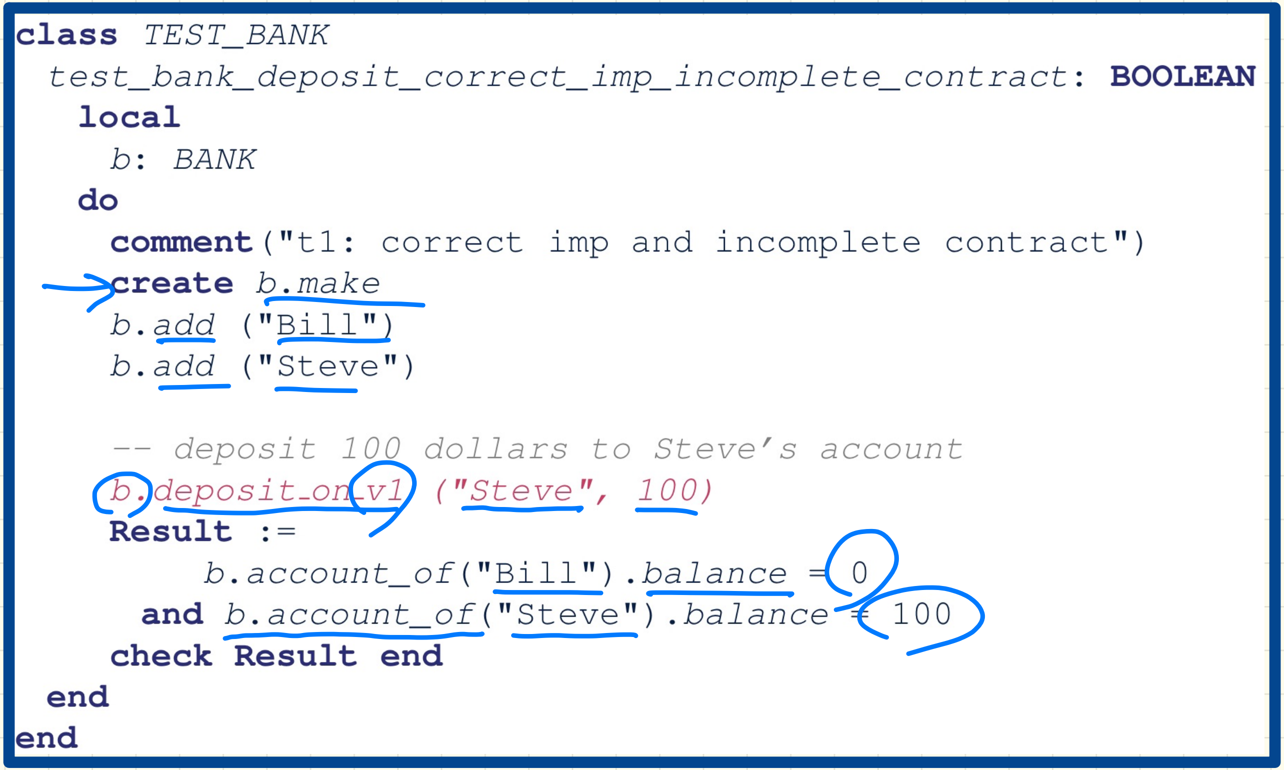
bank.account_of("Alan")

```
across accounts as acc
  some
    acc.owner ~ n
    acc.item.owner
  end
```



```
class Foo {

  m(..){
    x this = ..

  } x current :=

}
```

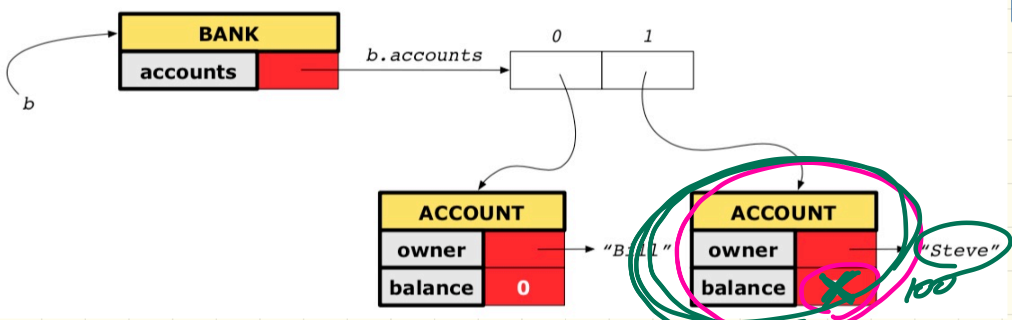
Unit Test for All 5 Versions



```
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)



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
```

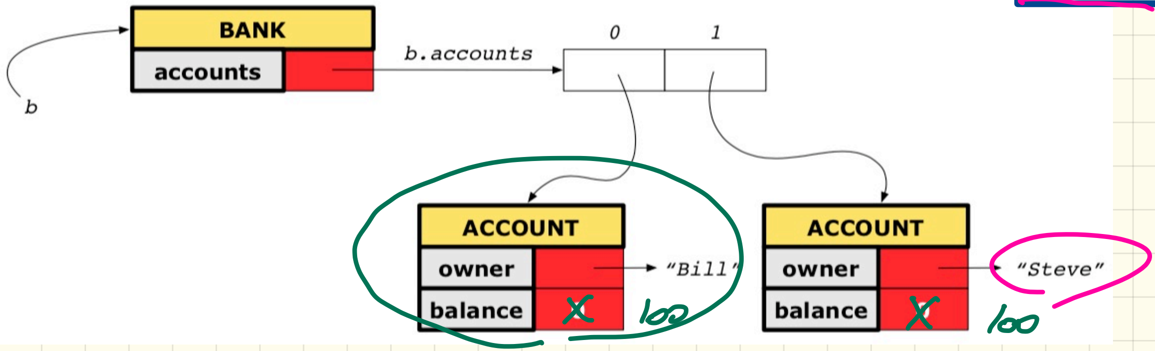
100 = 0 + 100

T

① old_b := Current.account_of(n).balance

Version 2: Incomplete Contracts, Wrong Implementation

b.deposit("Steve", 100)

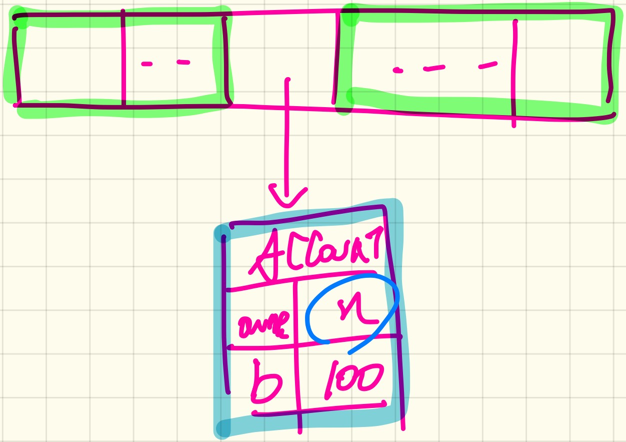


Steve 100

```
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
```

0

only concerns about owner n



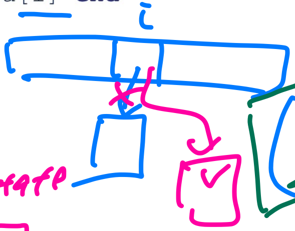
b.deposit_on(n, 50)

How to specify "all accounts except the one with owner n have remained the same"

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

Use of old in across Expression in Postcondition

```
class LINEAR_CONTAINER
create make
feature -- Attributes
  a: ARRAY[STRING]
feature -- Queries
  count: INTEGER do Result := a.count end
  get (i: INTEGER): STRING do Result := a[i] end
feature -- Commands
  make do create a.make_empty end
  update (i: INTEGER; v: STRING)
  do ...
  ensure -- Others Unchanged
    across
      1 |..| count as j
    all
      j.item /= i implies old get(j.item) ~ get(j.item)
    end
  end
end
```



only exist in post-state

(old get)(j.item) X

(old Current.deep_twin).get(j.item)

(old (Current)).get(j.item)

compiles

useless

c_v := get(j.item) → X j does not exist in pre-state

Hint: What value will be cached at runtime

before executing the implementation of update?