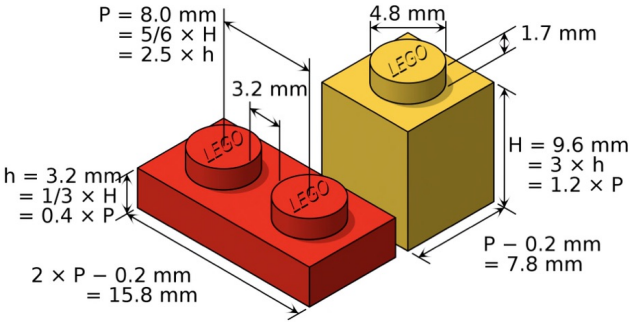
Lecture 2

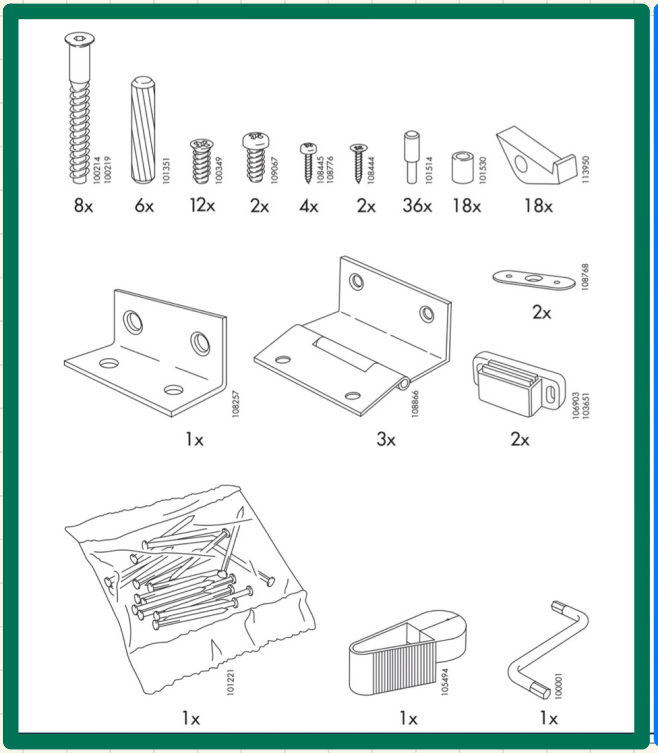
Part 1

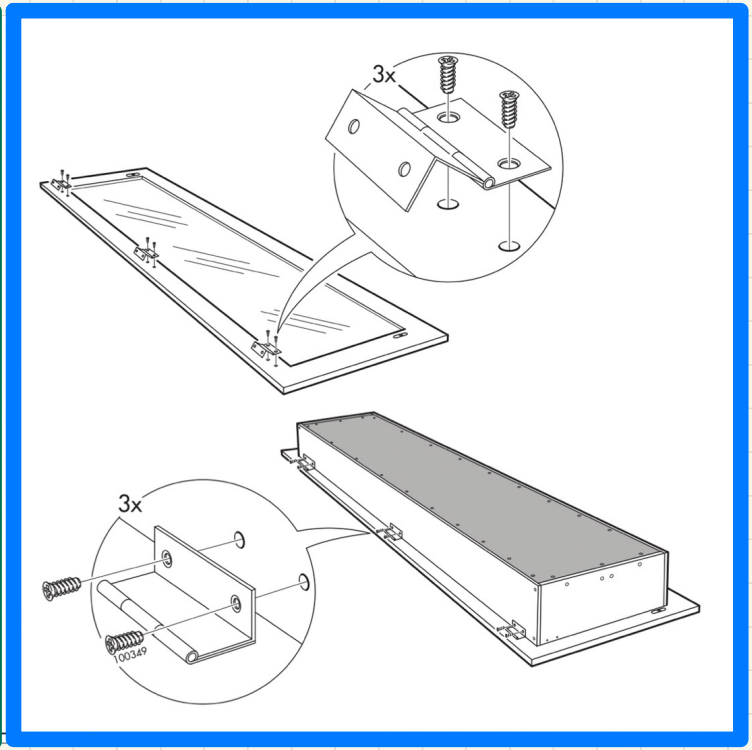
Modularity & Modular Design
Abstract Data Types (ADTs)

Modularity: Childhood Activities

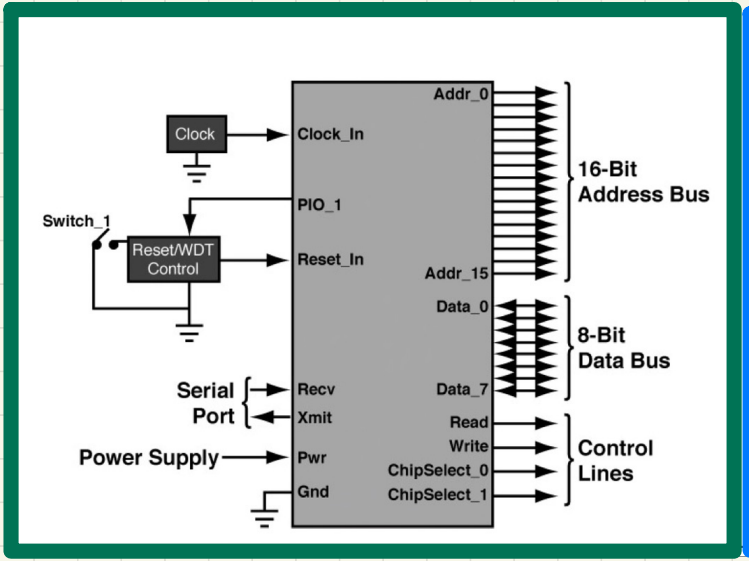


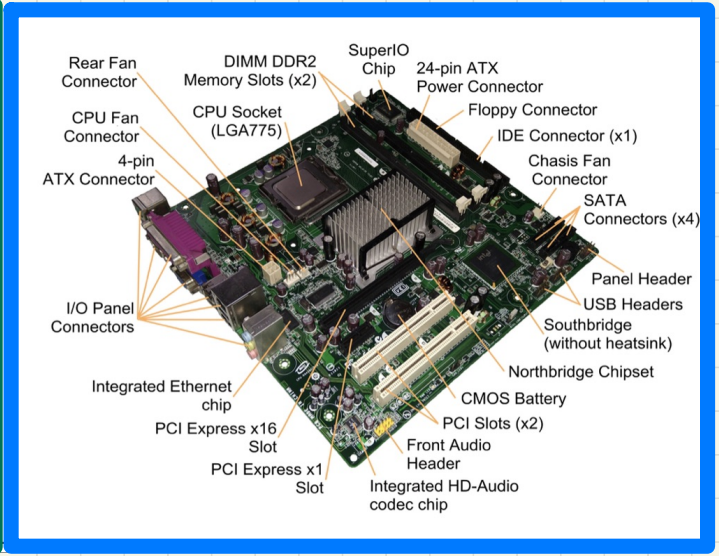
Modularity: Daily Constructions





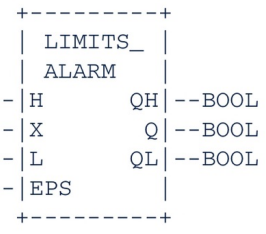
Modularity: Computer Architectures



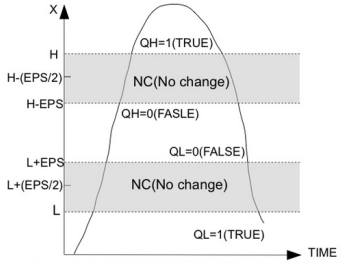


Modularity: System Developments

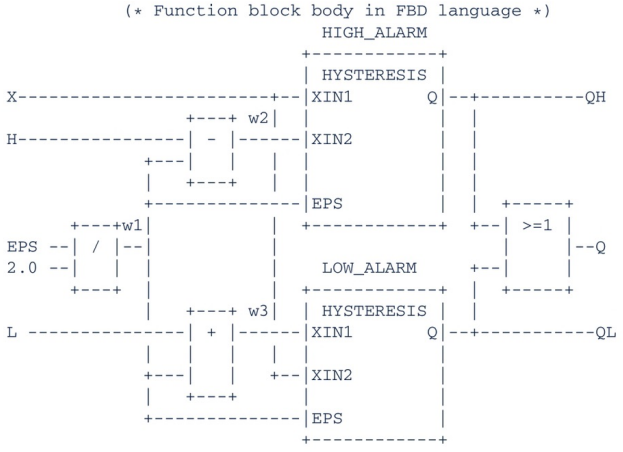
(* DECLARATION *)



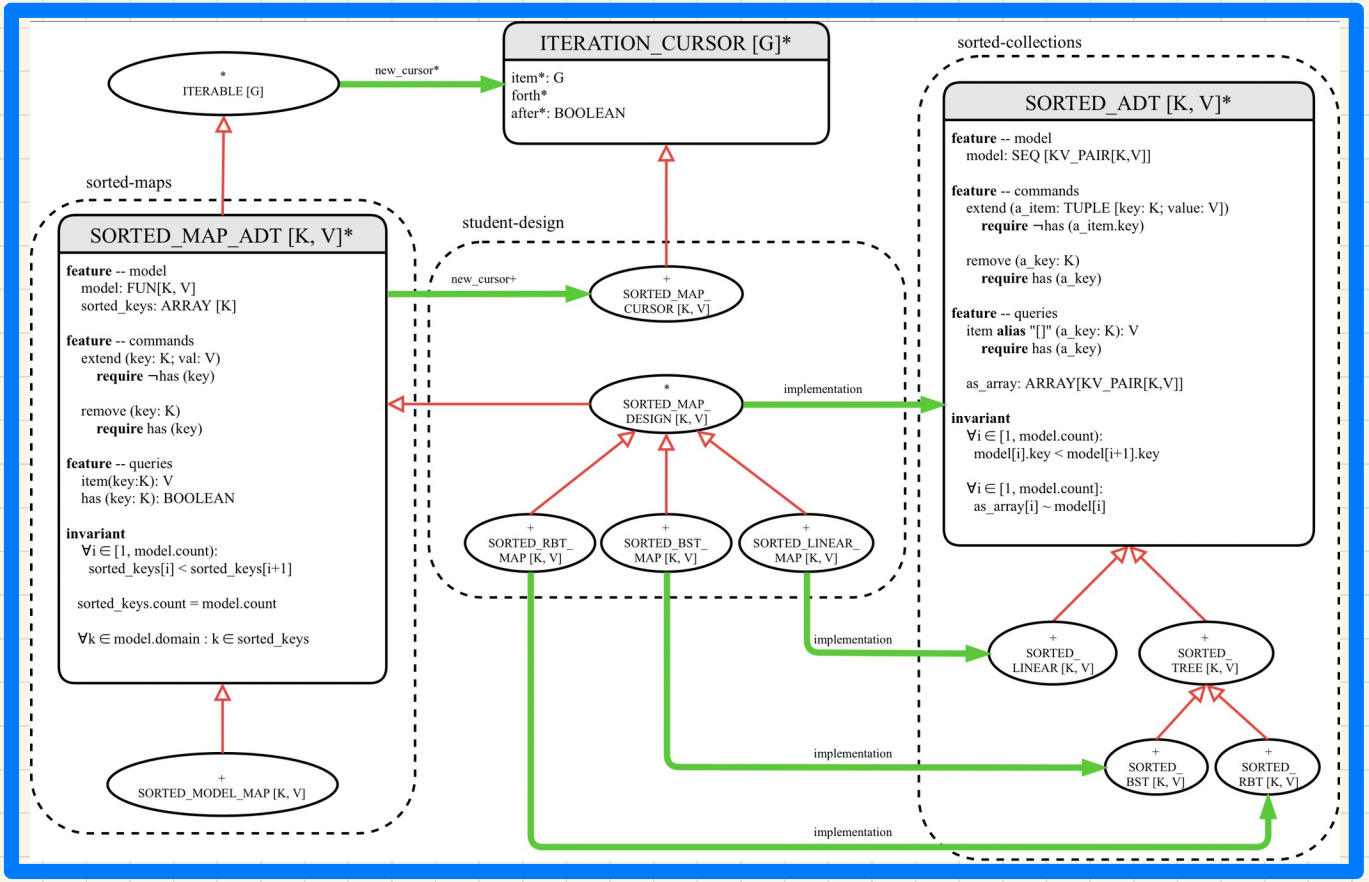

```
FUNCTION_BLOCK LIMITS_ALARM
  VAR_INPUT
    H   : REAL; (* High limit     *)
    X   : REAL; (* Variable value *)
    L   : REAL; (* Lower limit    *)
    EPS : REAL; (* Hysteresis     *)
  END_VAR
  VAR_OUTPUT
    QH : BOOL; (* High flag    *)
    Q  : BOOL; (* Alarm output *)
    QL : BOOL; (* Low flag     *)
  END_VAR
END_FUNCTION_BLOCK
```



(* Function block body in FBD language *)



Modularity: Software Design



Java Classes: Abstract Data Types?

E	**set**(int index, E element) Replaces the element at the specified position in this list with the specified element (optional operation).

set

```
E set(int index,
      E element)
```

Replaces the element at the specified position in this list with the specified element (optional operation).

Parameters:

index - index of the element to replace

element - element to be stored at the specified position

Returns:

the element previously at the specified position

Throws:

UnsupportedOperationException - if the set operation is not supported by this list

ClassCastException - if the class of the specified element prevents it from being added to this list

NullPointerException - if the specified element is null and this list does not permit null elements

IllegalArgumentException - if some property of the specified element prevents it from being added to this list

IndexOutOfBoundsException - if the index is out of range (index < 0 || index >= size())

Interface List<E>

Type Parameters:

E - the type of elements in this list

All Superinterfaces:

Collection<E>, Iterable<E>

All Known Implementing Classes:

AbstractList, AbstractSequentialList, ArrayList, AttributeList, CopyOnWriteArrayList, LinkedList, RoleList, RoleUnresolvedList, Stack, Vector

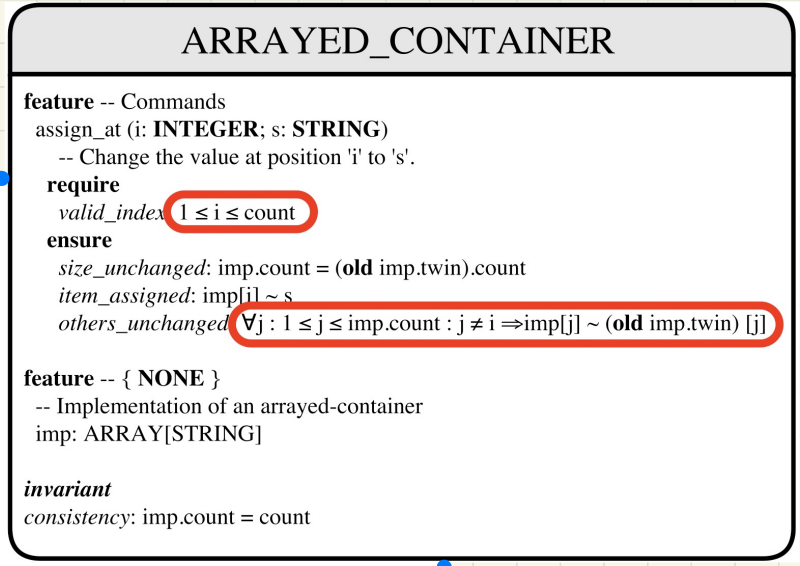
---

```
public interface List<E>
extends Collection<E>
```

An ordered collection (also known as a *sequence*). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list.

Eiffel Classes: Abstract Data Types?

Design Diagram



Contract View

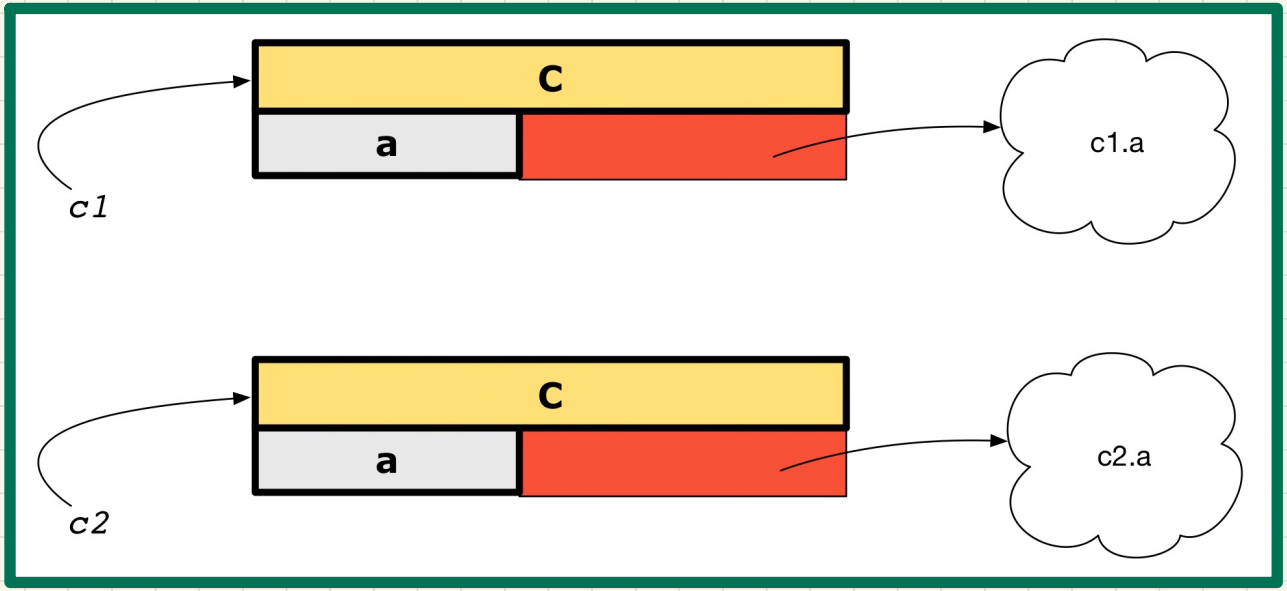
```
class interface ARRAYED_CONTAINER
feature -- Commands
  assign_at (i: INTEGER; s: STRING)
      -- Change the value at position 'i'
    require
      valid_index: 1 <= i and i <= count
    ensure
      size_unchanged:
        imp.count = (old imp.twin).count
      item_assigned:
        imp [i] ~ s
      others_unchanged:
        across
          1 |..| imp.count as j
        all
          j.item /= i implies imp [j.item] ~ (old imp.twin) [j.item]
        end
  count: INTEGER
invariant
  consistency: imp.count = count
end -- class ARRAYED_CONTAINER
```

Lecture 2

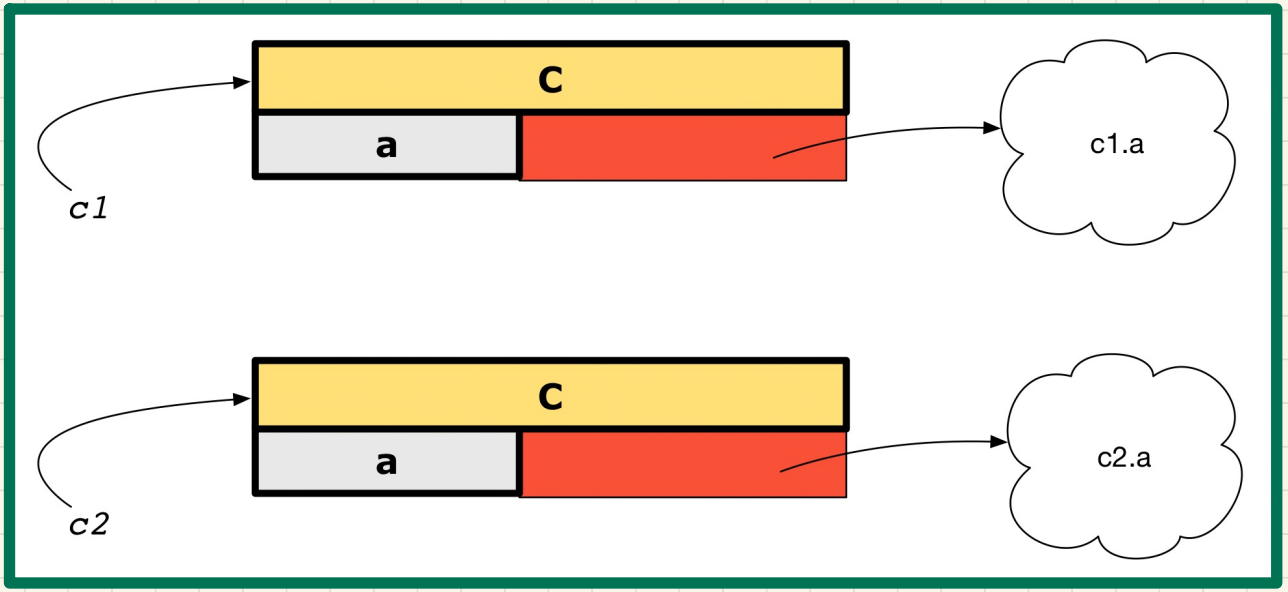
Part 2

Copying Objects:
Reference vs. Shallow vs. Deep

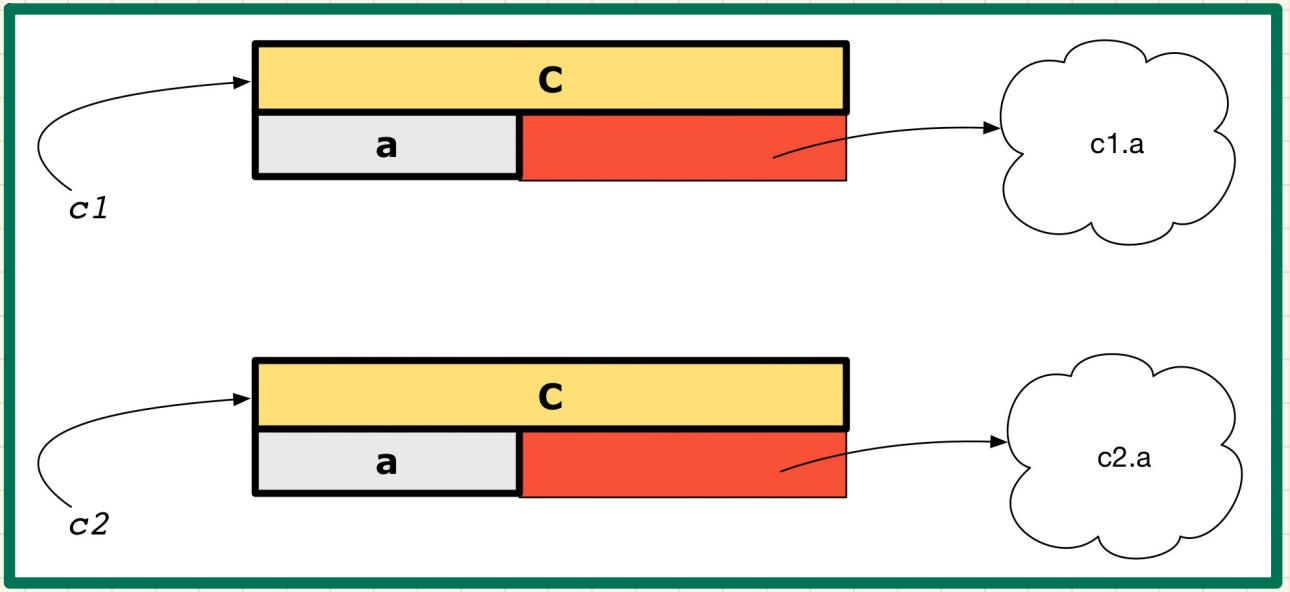
Reference Copy: c1 := c2



Shallow Copy: c1 := c2.twin



Deep Copy: c1 := c2.deep_twin



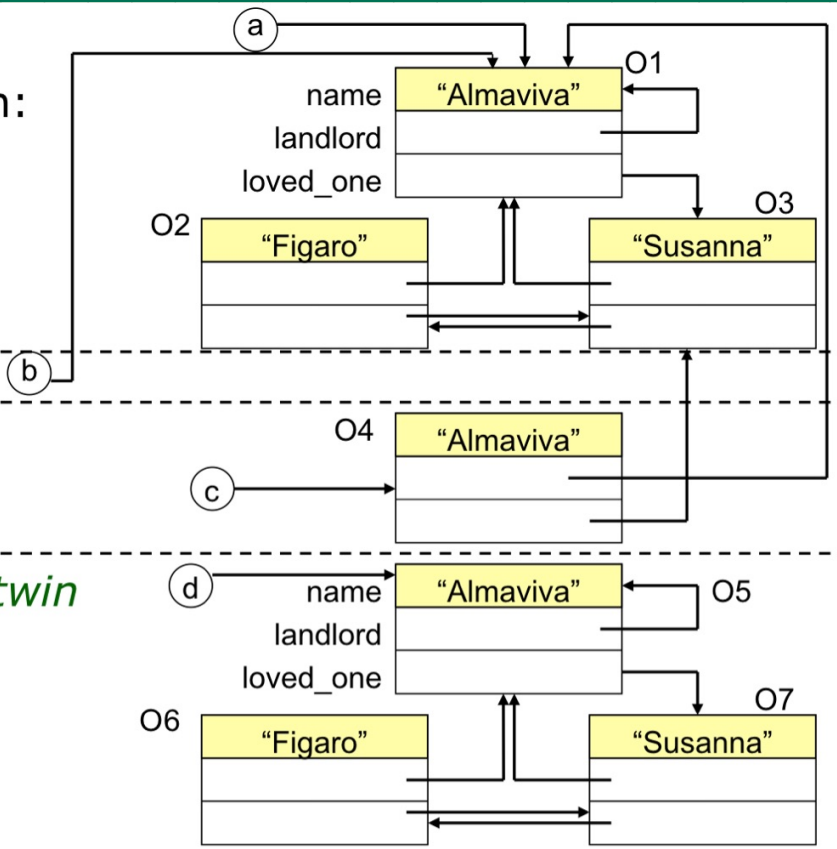
Reference vs. Shallow vs. Deep Copies

- Initial situation:
- Result of:

b := *a*

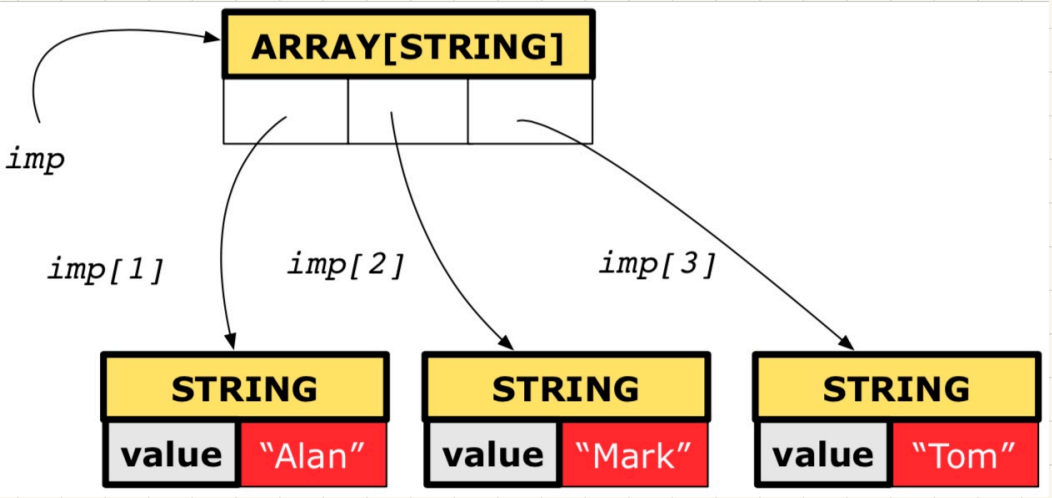
c := *a.twin*

d := *a.deep_twin*



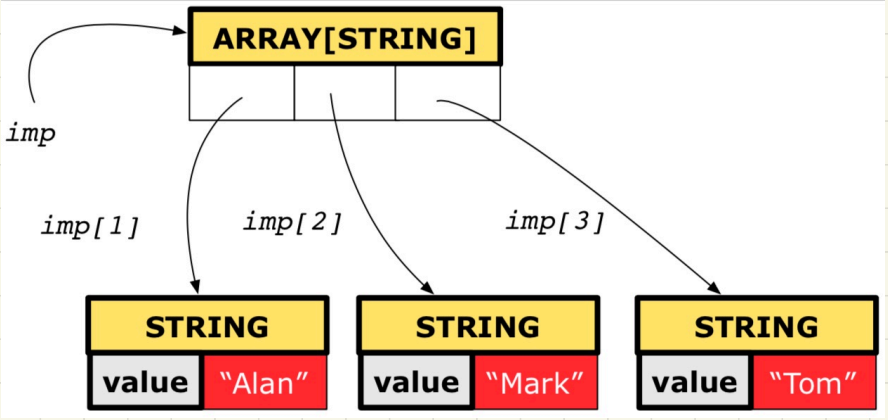
Collection Objects: Reference Copy & Make Changes

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



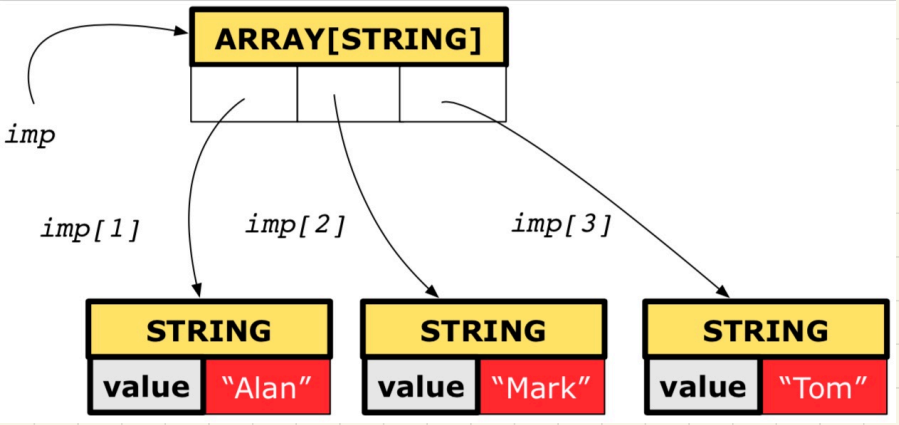
Collection Objects: Shallow Copy & Make 1st-Level Changes

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



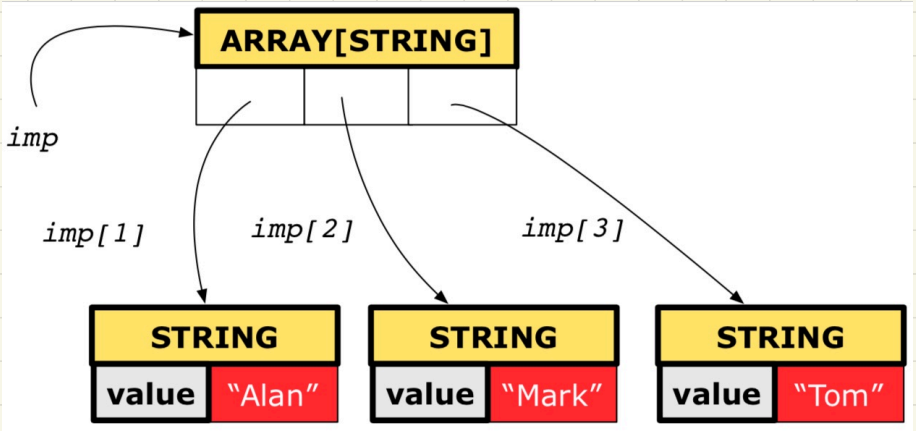
Collection Objects: Shallow Copy & Make 2nd-Level Changes

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



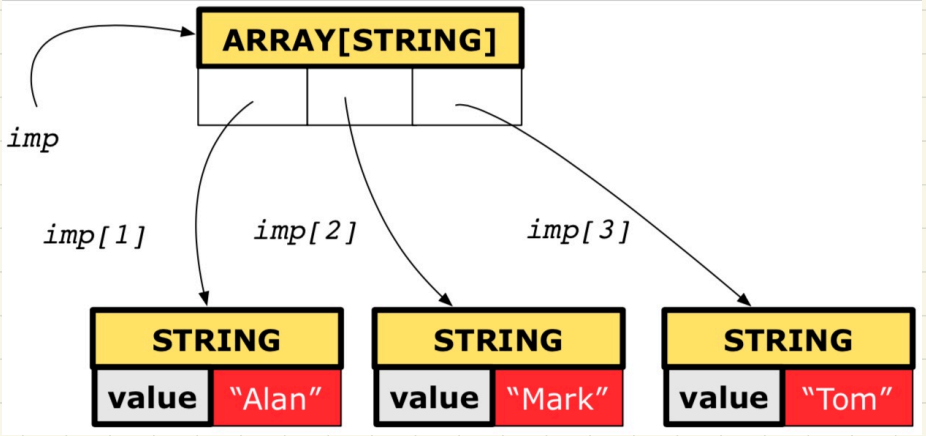
Collection Objects: Deep Copy & Make 1st-Level Changes

```
1  old_imp := imp.deep_twin
2  Result := old_imp = imp   -- Result = [redacted]
3  imp[2] := "Jim"
4  Result :=
5    across 1 |..| imp.count is j
6    all imp [j] ~ old_imp [j] end -- Result = [redacted]
```



Collection Objects: Deep Copy & Make 2nd-Level Changes

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



Lecture 2

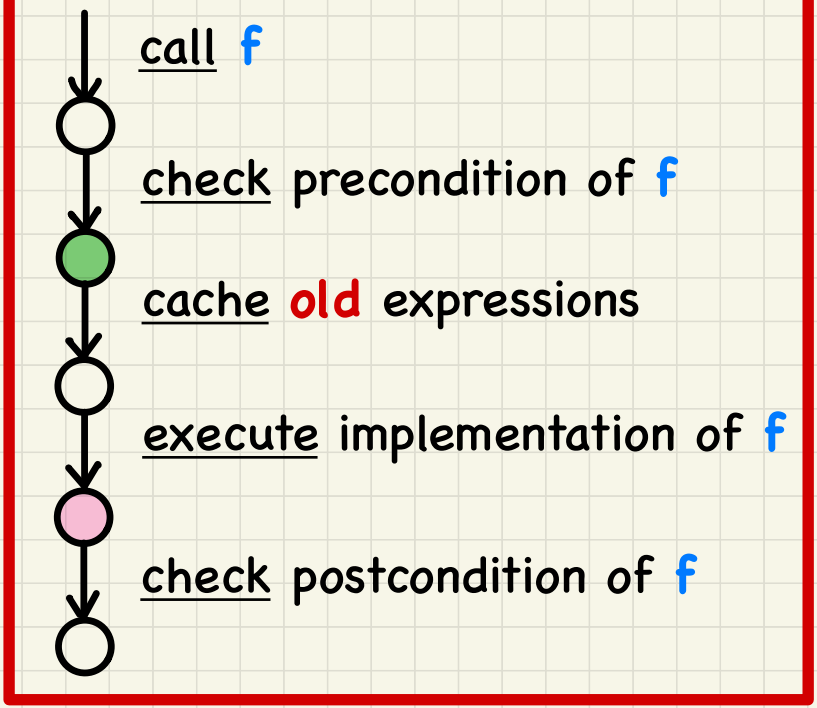
Part 3

Writing Complete Postconditions

Contract View

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

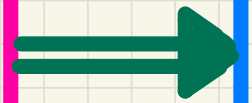
Runtime Contract Checks



Caching Values for old Expressions in Postconditions

```
class BANK
  accounts: ARRAY[ACCOUNT]

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



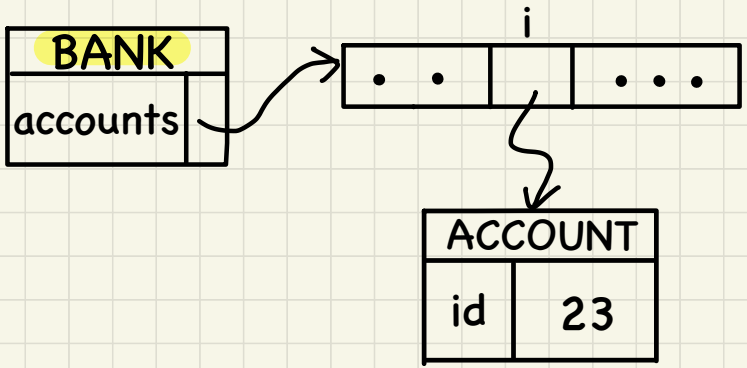
```
class ACCOUNT
  id: INTEGER
end
```

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

How to cache at runtime?



Revisit: Bank Accounts in Java V5

```
1  public class AccountV5 {
2    public void withdraw(int amount) throws
3       WithdrawAmountNegativeException, WithdrawAmountTooLargeException {
4      int oldBalance = this.balance;
5      if(amount < 0) { /* negated precondition */
6        throw new WithdrawAmountNegativeException(); }
7      else if (balance < amount) { /* negated precondition */
8        throw new WithdrawAmountTooLargeException(); }
9      else { this.balance = this.balance - amount; }
10     assert this.getBalance() > 0 :"Invariant: positive balance";
11     assert this.getBalance() == oldBalance - amount :
12       "Postcondition:  balance deducted"; }
```

How does the corresponding Eiffel design look like (with automatic caching of pre-state values)?

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
```

Hint: What value will be cached at runtime

before executing the implementation of update?

```
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
```

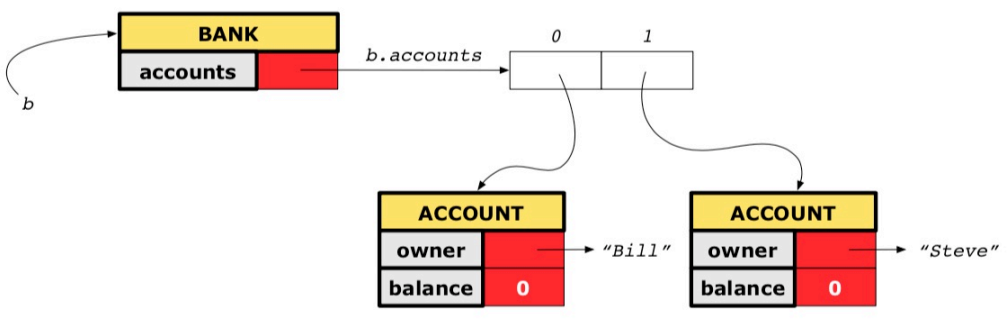
Unit Test for All 5 Versions

```
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

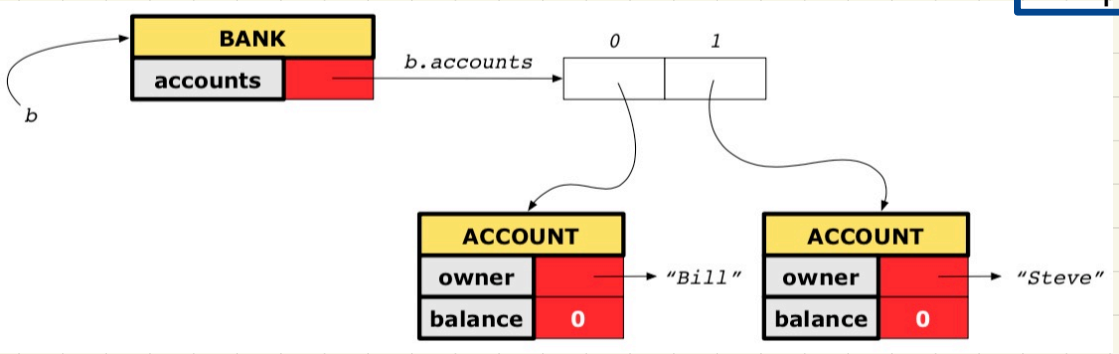
b.deposit("Steve", 100)




```
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
```

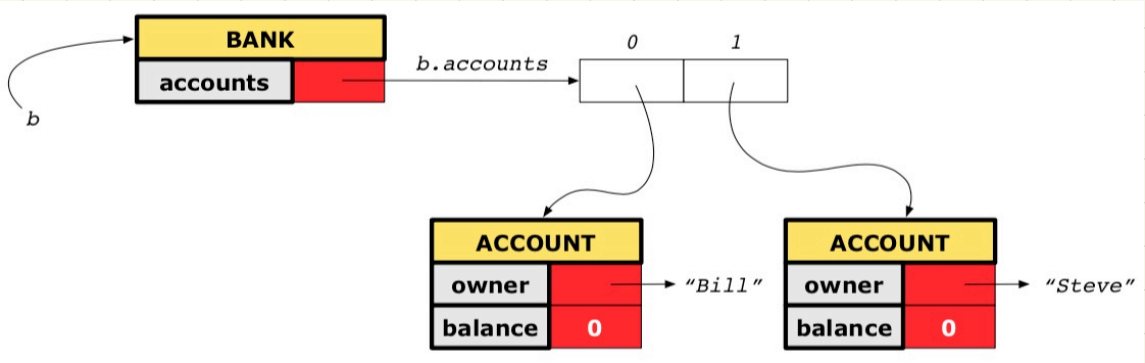
Version 2: Incomplete Contracts, Wrong Implementation

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
```

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



b.deposit("Steve", 100)

1st Iteration

acc.owner /~ n **implies** acc ~ **Current**.account_of (acc.owner)

2nd Iteration

acc.owner /~ n **implies** acc ~ **Current**.account_of (acc.owner)

```
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
```

Use of across in Postcondition

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

For each iteration:

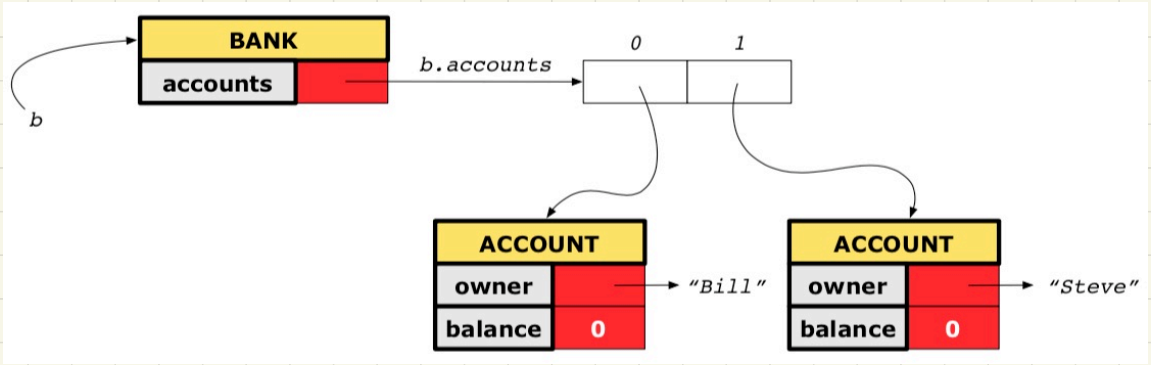
Case 1: acc.owner is not n

```
acc.owner /~ n implies acc ~ Current.account_of (acc.owner)
```

Case 2: acc.owner is n

```
acc.owner /~ n implies acc ~ Current.account_of (acc.owner)
```

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



b.deposit("Steve", 100)

1st Iteration

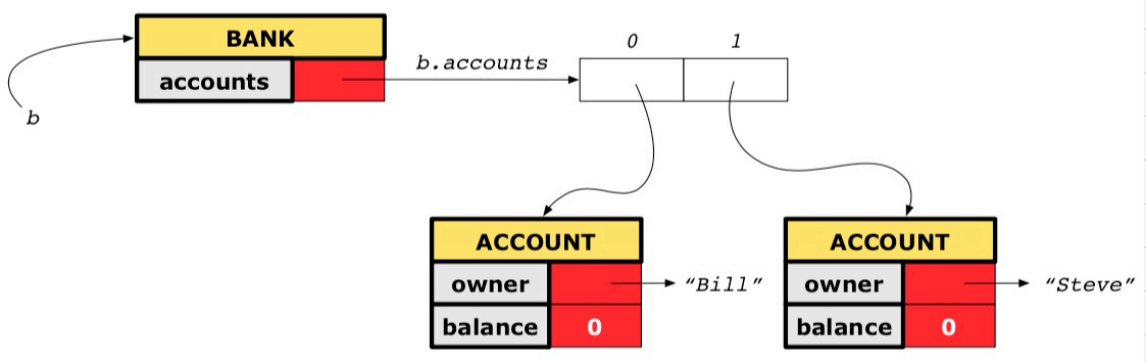
acc.owner /~ n **implies** acc ~ **Current**.account_of (acc.owner)

2nd Iteration

acc.owner /~ n **implies** acc ~ **Current**.account_of (acc.owner)

```
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)

1st Iteration

acc.owner /~ n **implies** acc ~ **Current**.account_of (acc.owner)

2nd Iteration

acc.owner /~ n **implies** acc ~ **Current**.account_of (acc.owner)

```
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
```

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`