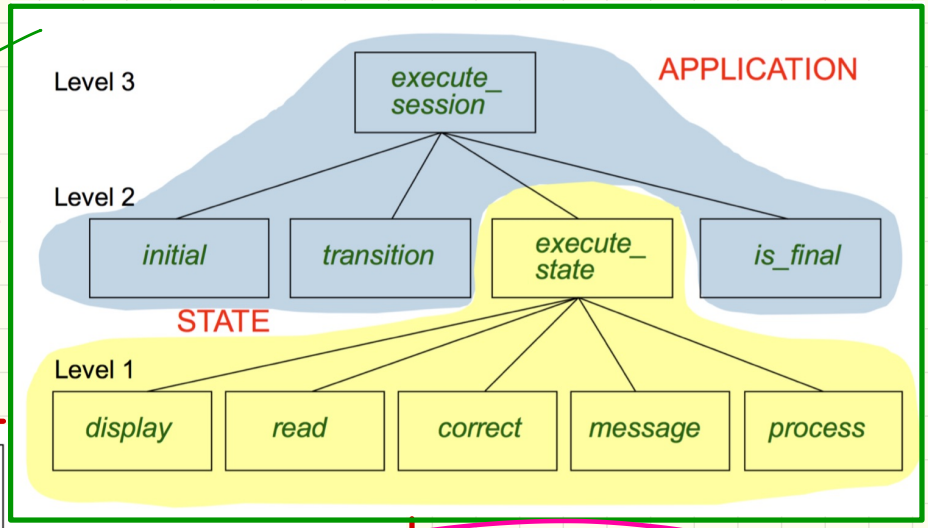
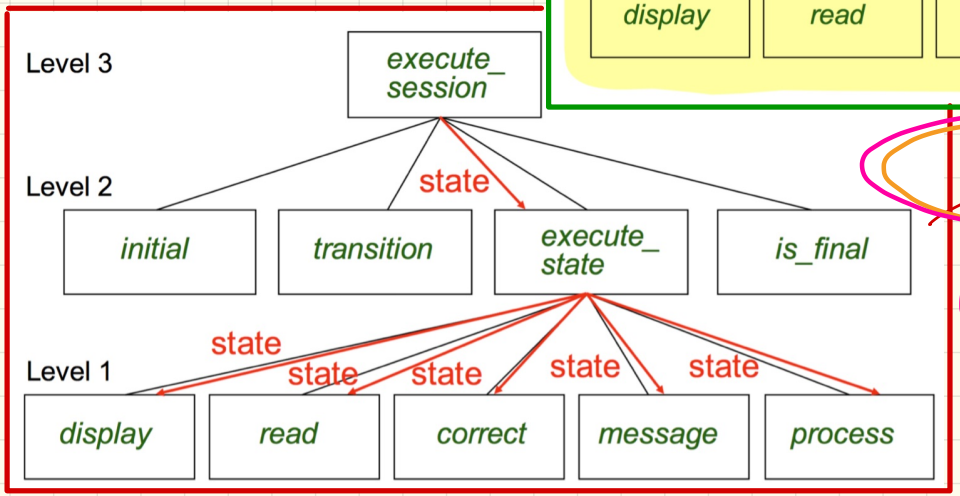
Lecture 19
Monday March 16

Moving from Top-Down Design to OO Design

Object-Oriented

current_state: **STATE**
current_state.*execute_session*



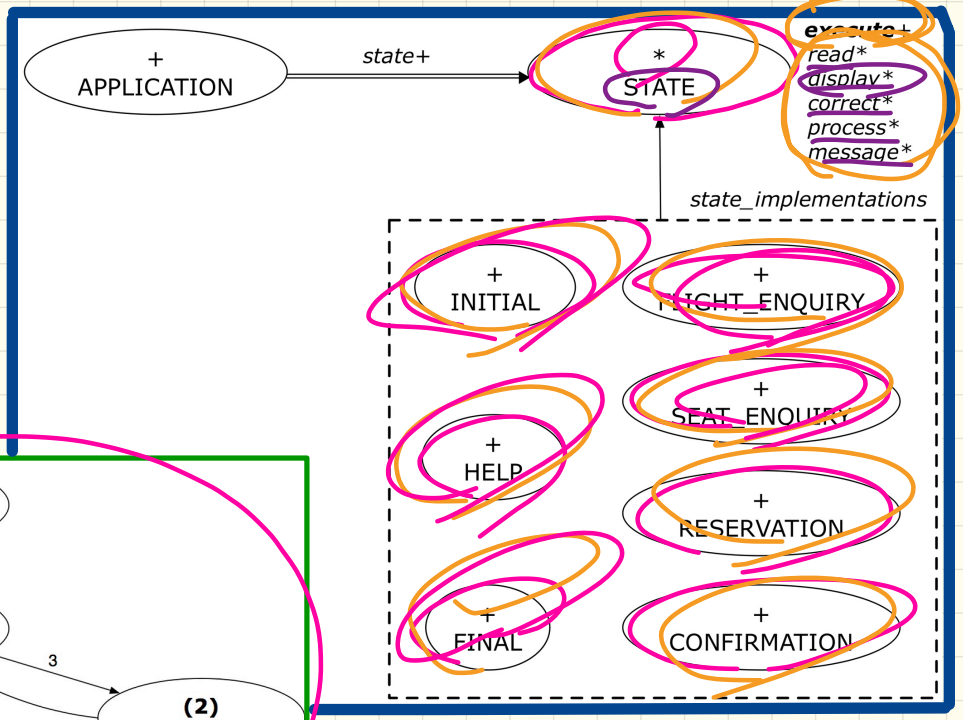


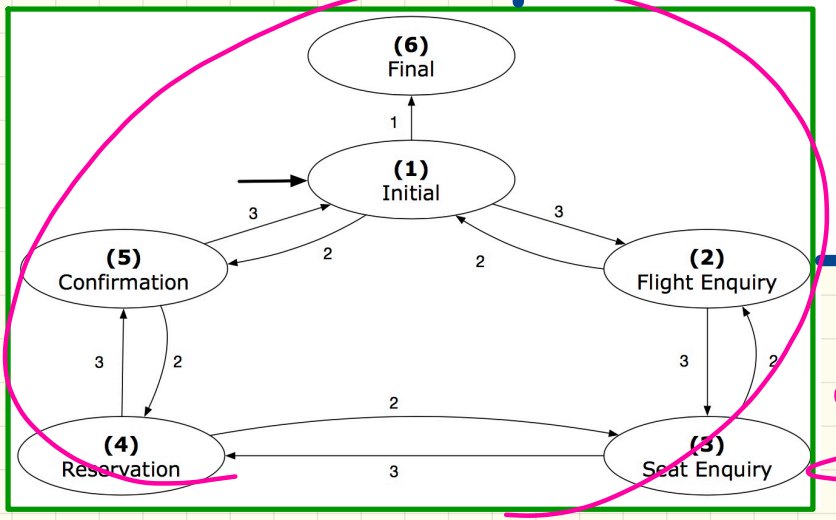
Top-Down

current_state: **INTEGER**
execute_session(current_stste)

State Pattern: Architecture

template






```
s: STATE
create {SEAT_ENQUIRY} s.make
s.execute
create {CONFIRMATION} s.make
s.execute
```

State Pattern: State Module

```
deferred class STATE
  read
    -- Read user's inputs
    -- Set 'answer' and 'choice'
    deferred end
  answer: ANSWER
    -- Answer for current state
  choice: INTEGER
    -- Choice for next step
  display
    -- Display current state
    deferred end
  correct: BOOLEAN
    deferred end
  process
    require correct
    deferred end
  message
    require not correct
    deferred end
```

template

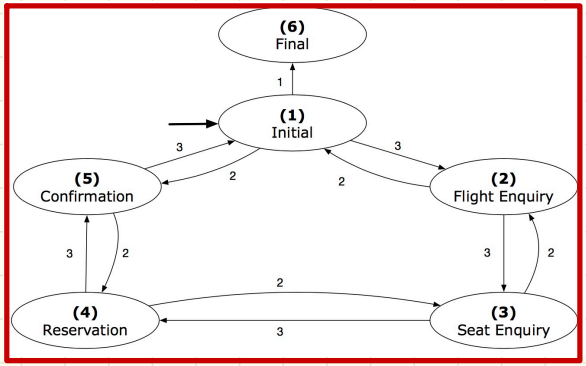
```
  execute
    local
      good: BOOLEAN
    do
      from
      until
        good
      loop
        display
        -- Set answer and choice
        read
        good := correct
        if not good then
          message
        end
      end
      process
    end
end
```

TEMPLATE

```
s: STATE
create {SEAT_ENQUIRY} s.make
s.execute
create {CONFIRMATION} s.make
s.execute
```

State Pattern: Application Module

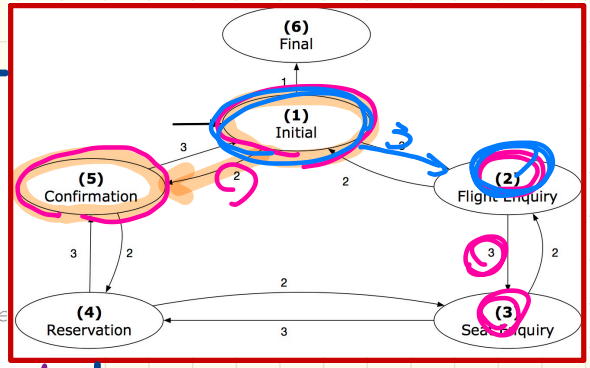
```
class APPLICATION create make
feature {NONE} -- Implementation of Transition Graph
  transition: ARRAY2[INTEGER]
    -- State transitions: transition[state, choice]
  states: ARRAY[STATE]
    -- State for each index, constrained by size of 'transition'
feature
  initial: INTEGER
  number_of_states: INTEGER
  number_of_choices: INTEGER
  make(n, m: INTEGER)
    do number_of_states := n
       number_of_choices := m
       create transition.make_filled(0, n, m)
       create states.make_empty
    end
feature
  put_state(s: STATE; index: INTEGER)
    require 1 ≤ index ≤ number_of_states
    do states.force(s, index) end
  choose_initial(index: INTEGER)
    require 1 ≤ index ≤ number_of_states
    do initial := index end
  put_transition(tar, src, choice: INTEGER)
    require
      1 ≤ src ≤ number_of_states
      1 ≤ tar ≤ number_of_states
      1 ≤ choice ≤ number_of_choices
    do
      transition.put(tar, src, choice)
    end
invariant
  transition.height = number_of_states
  transition.width = number_of_choices
end
```

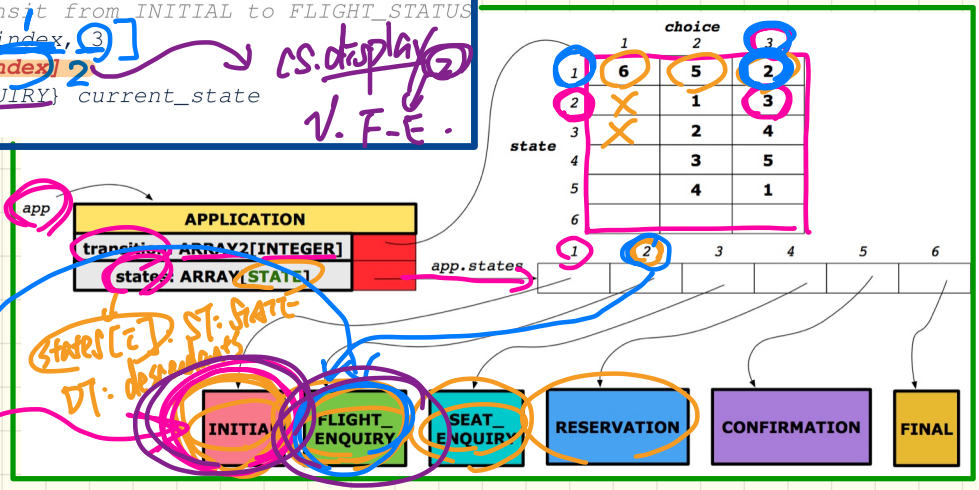


State Pattern: Test

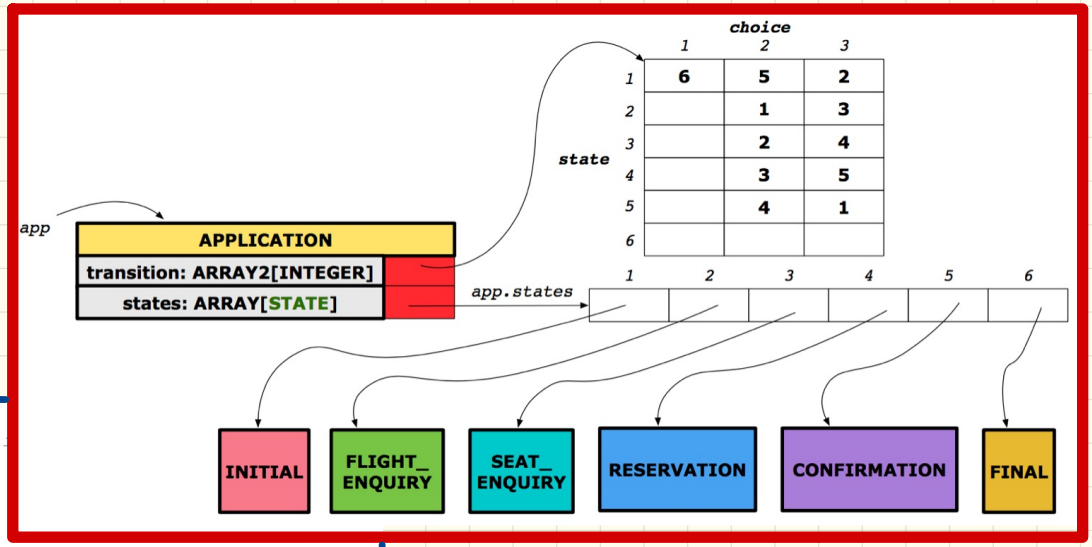
```
test_application: BOOLEAN
  local
    app: APPLICATION ; current_state: STATE ; index: INTEGER
  do
    create app.make (6, 3)
    app.put_state (create {INITIAL}.make, 1)
    -- Similarly for other 5 states.
    app.choose_initial (1)
    -- Transit to FINAL given current state INITIAL and choice
    app.put_transition (6, 1, 1)
    -- Similarly for other 10 transitions.

    index := app.initial
    current_state := app.states [index]
    Result := attached {INITIAL} current_state
    check Result end
    -- Say user's choice is 3: transit from INITIAL to FLIGHT_STATUS
    index := app.transition.item (index, 3)
    current_state := app.states [index]
    Result := attached {FLIGHT_ENQUIRY} current_state
end
```





State Pattern: Interactive Session




```
class APPLICATION
feature {NONE} -- Implementati
  transition: ARRAY2[INTEGER]
  states: ARRAY[STATE]
feature
  execute_session
    local
      current_state: STATE
      index: INTEGER
    do
      from
        index := initial
      until
        is_final (index)
      loop
        current_state := states[index] -- polymorphism
        current_state.execute -- dynamic binding
        index := transition.item (index, current_state.choice)
      end
    end
end
```

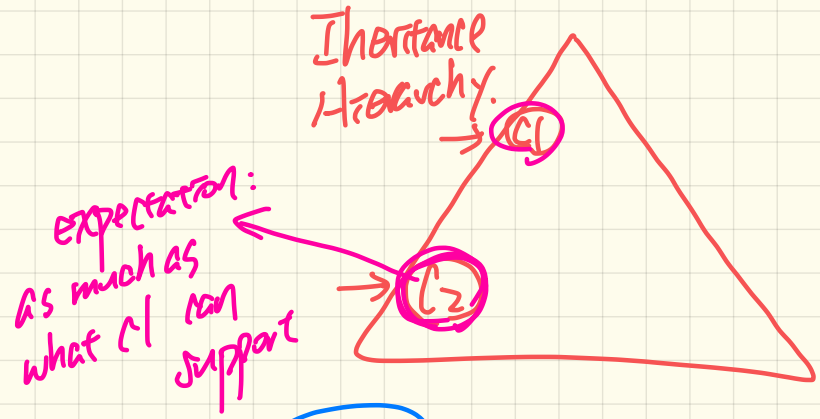
ST: STATE

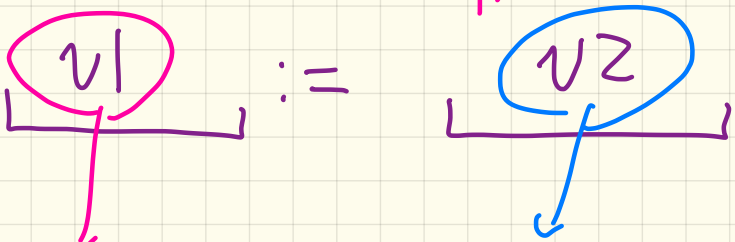
compile?

ST: STATE

D.B.: depending on the DT of the curr. C.S.'s version of EXECUTE will be called

Polymorphism





Compile?

ST: ?

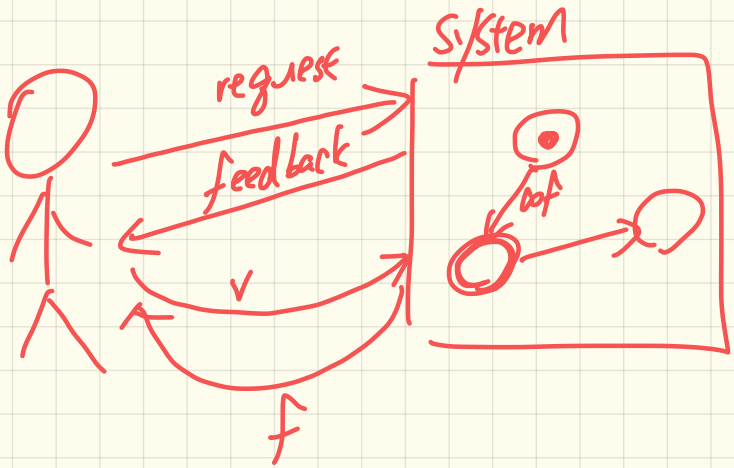
C1

ST: ?

C2

a descendant of

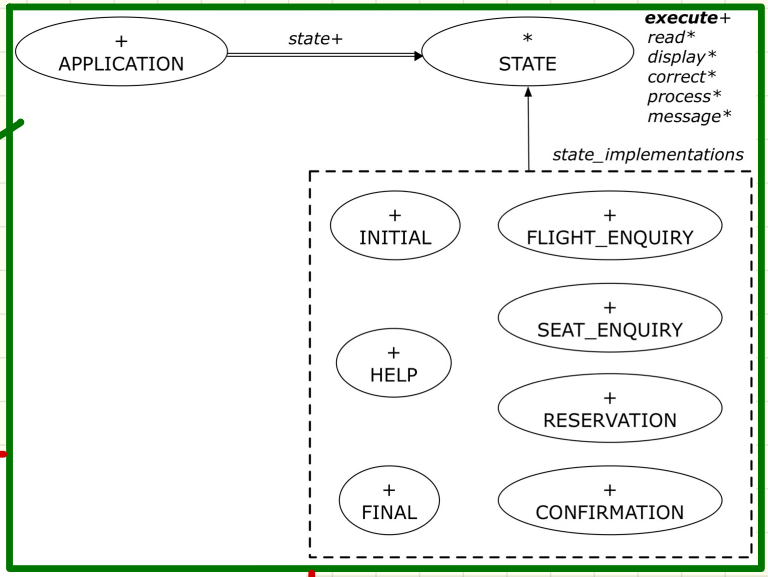
ETF -1

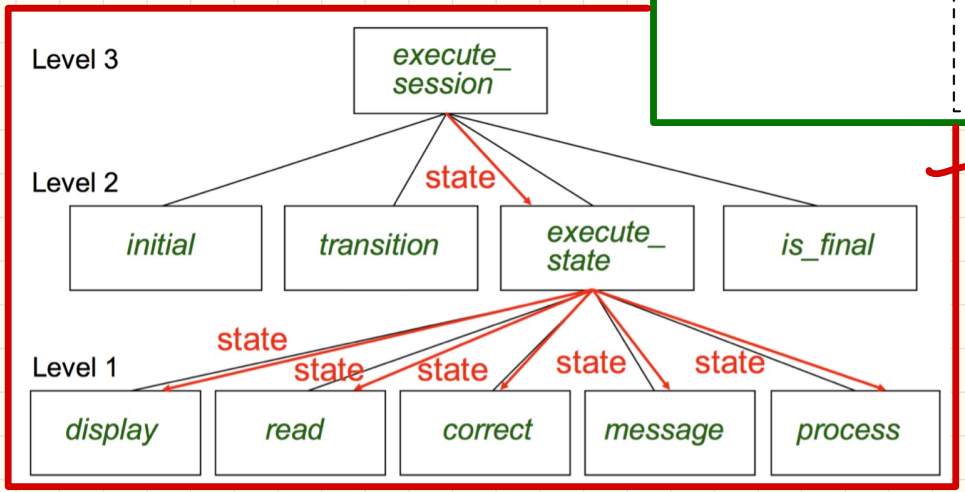


Interactive System: Top-Down Design vs. OO Design

Object-Oriented

current_state: **STATE**
current_state.*execute_session*





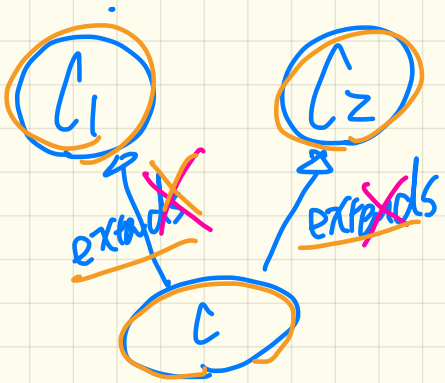
Top-Down

current_state: **INTEGER**
execute_session(current_stste)

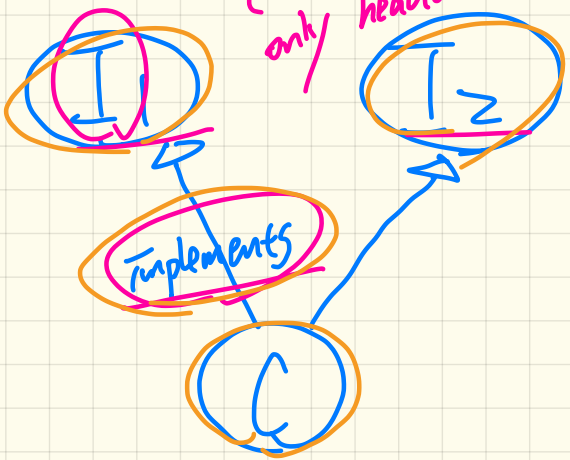
Java

M.I. limited to interface-

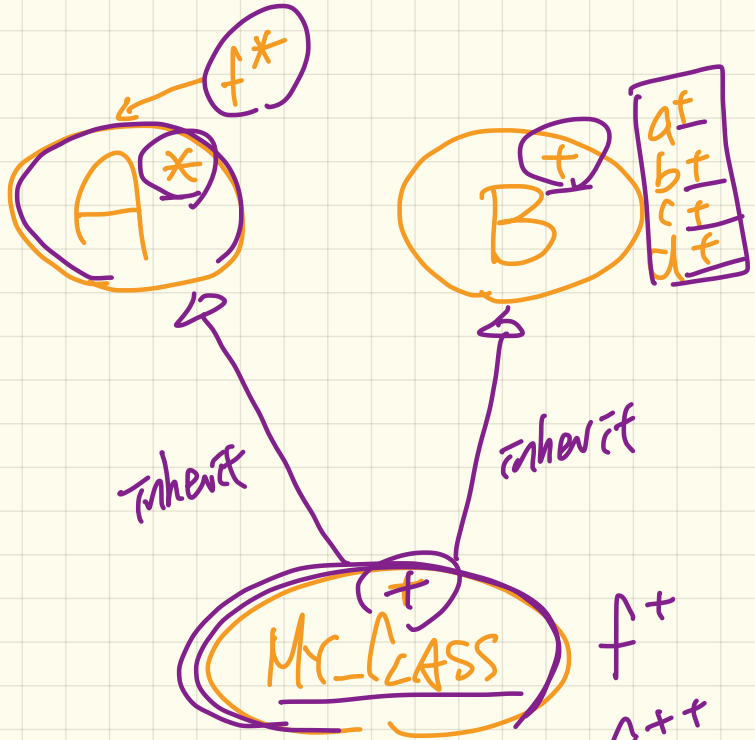
class C extends C1, C2



interfaces (no implementation only headers are inherited



Eiffel.



f++

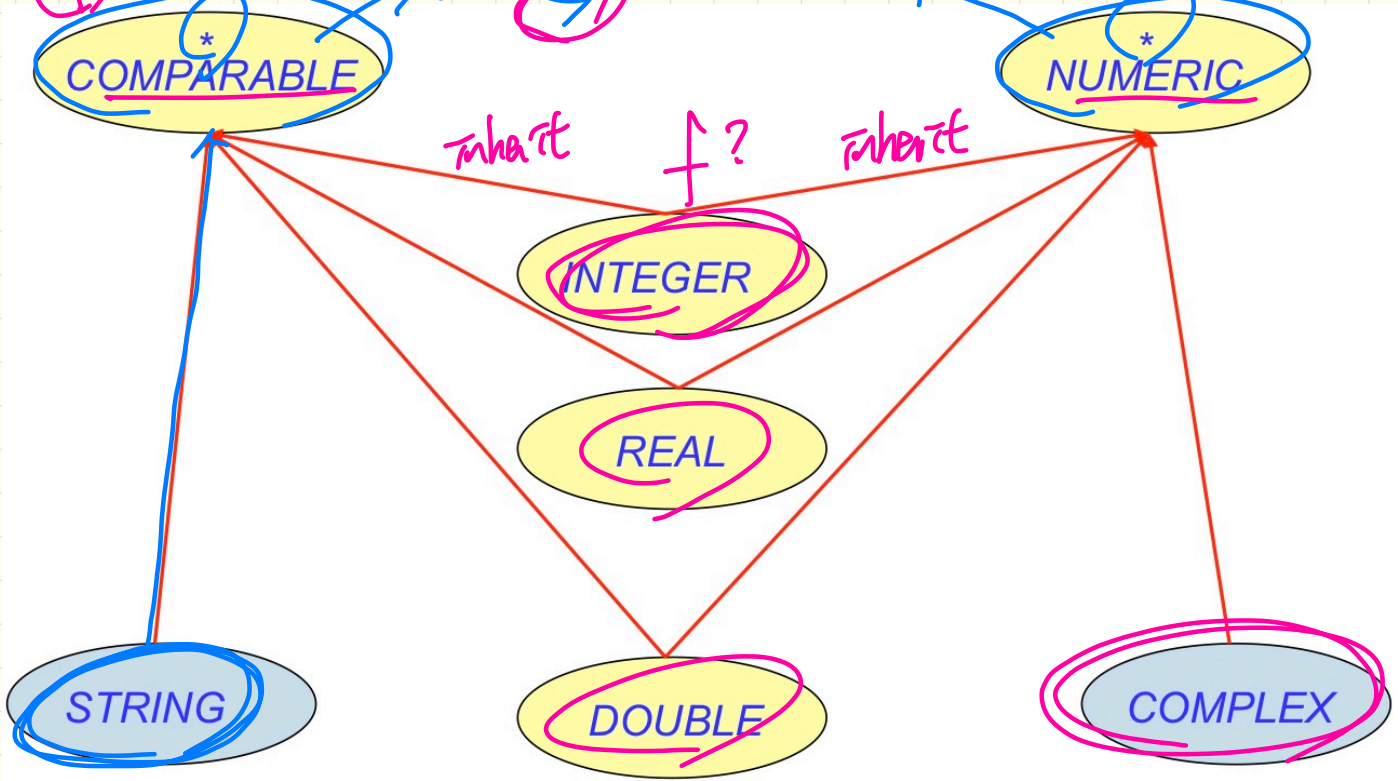
a+++

b+++

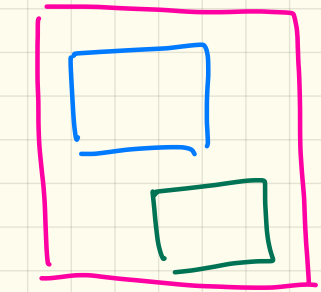
c+++

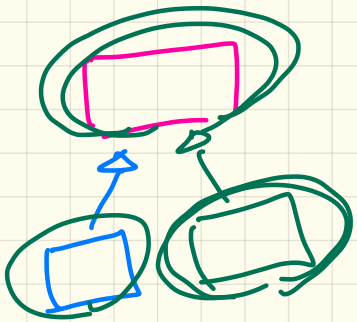
d+++

Multiple Inheritance: Example



Design 1





superman class

class

WINDOW

feature

xpos, ypos

width, height

feature

move, add_child

parents, children

End

dimension of win.

tree of wid.

W

Multiple Inheritance

Single Choice Principle

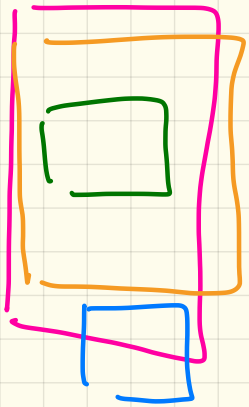
Cohesion violated

- parents and children are included in the same class.

Multiple Inheritance: Exercise

```
class RECTANGLE
  feature -- Queries
    width, height: REAL
    xpos, ypos: REAL
  feature -- Commands
    make (w, h: REAL)
    change_width
    change_height
    move
end
```

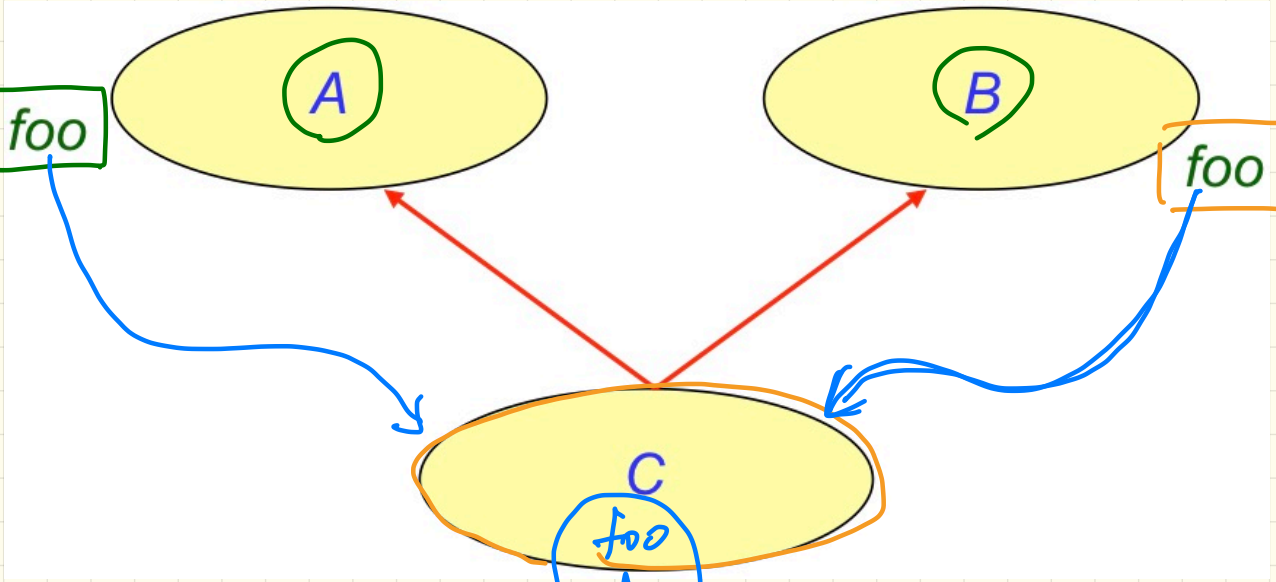
```
class TREE[G]
  feature -- Queries
    descendants: ITERABLE[G]
  feature -- Commands
    add (c: G)
        -- Add a child 'c'.
end
```



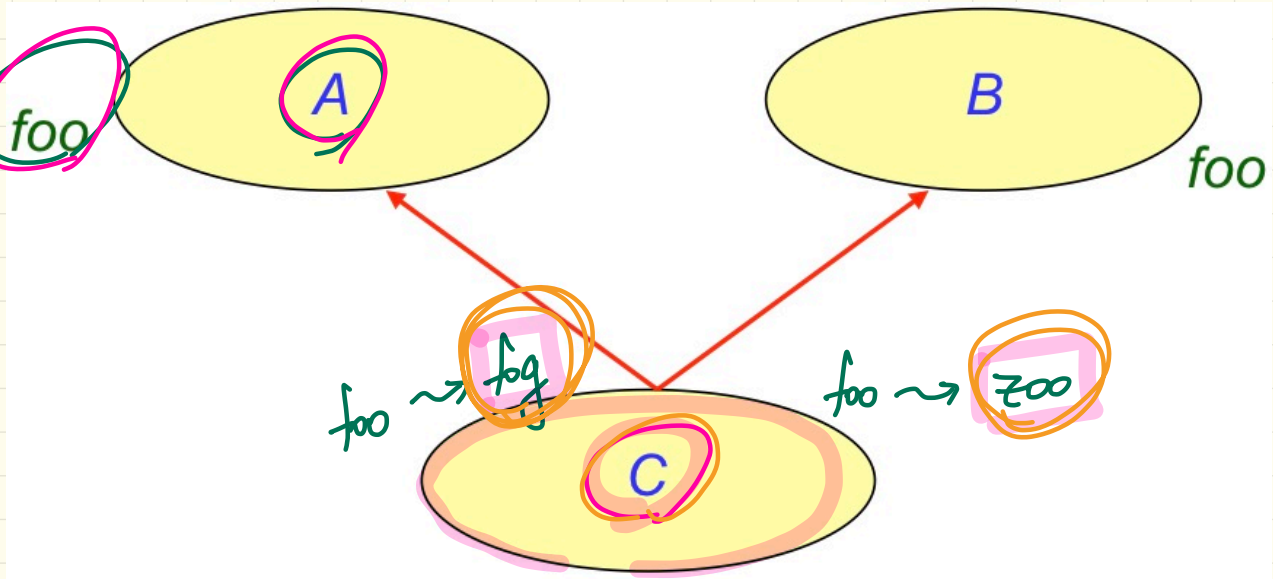
```
class WINDOW
  inherit
    RECTANGLE
    TREE[WINDOW]
end
```

```
test_window: BOOLEAN
  local w1, w2, w3, w4: WINDOW
  do
    create w1.make(8, 6) ; create w2.make(4, 3)
    create w3.make(1, 1) ; create w4.make(1, 1)
    w2.add(w4) ; w1.add(w2) ; w1.add(w3)
    Result := w1.descendants.count = 2
  end
```

Multiple Inheritance: Name Clashes



class C
inherit
A
B



obj1 : A

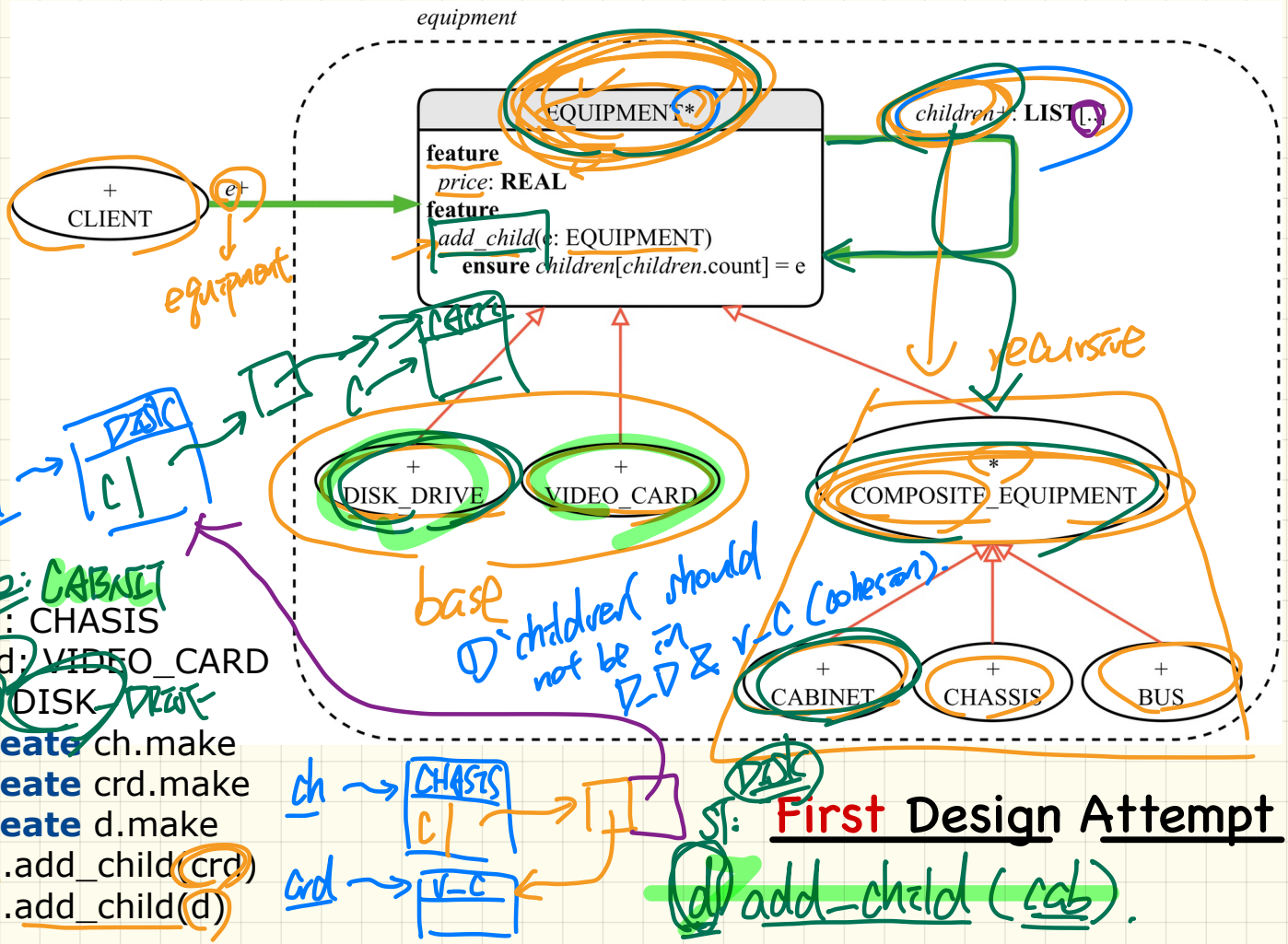
① obj1.foo ✓
↳ ST: A

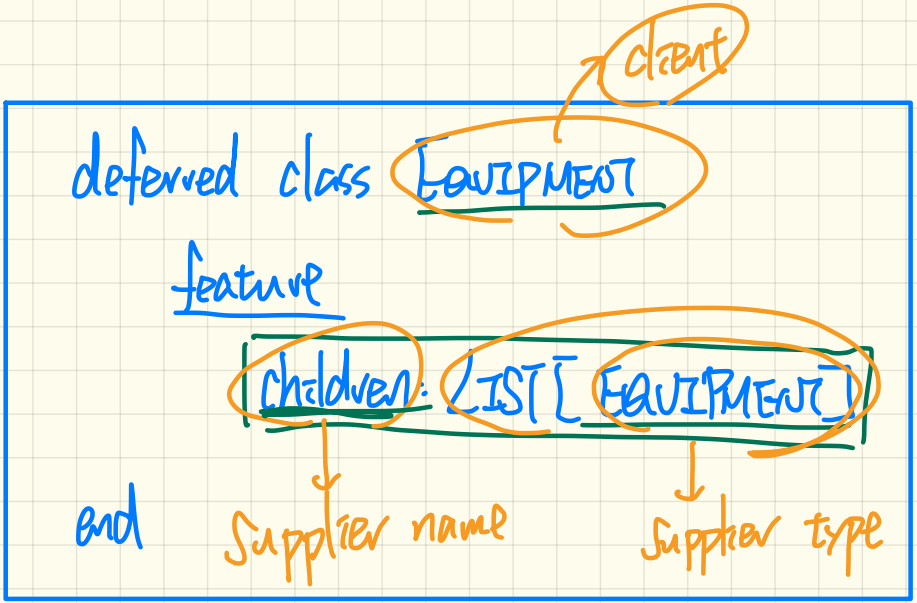
② obj1.foo ✗
↳ ST: A

obj2 : C

③ obj2.foo ✗
↳ ST: C

④ obj2.foo ✓


equipment
EQUIPMENT*
feature
price: REAL
feature
add_child(e: EQUIPMENT)
ensure children[children.count] = e
children+: LIST[..]
+
CLIENT
e+
equipment
+
DISK_DRIVE
+
VIDEO_CARD
*
COMPOSITE_EQUIPMENT
+
CABINET
+
CHASSIS
+
BUS
recursive
base
d
c
crd
cab: CABINET
ch: CHASIS
crd: VIDEO_CARD
d: DISK_DRIVE
create ch.make
create crd.make
create d.make
ch.add_child(crd)
ch.add_child(d)
ch
crd
s:
d.add_child(cab).
First Design Attempt




```
class Node {
    Node next;
}
```