Monday March 4

Lecture 14

Lab Test 2

- ETF
- Undo/Redo (OOSC2 ch. 21)

General Book

Supplier

```
class BOOK
  names: ARRAY[STRING]
  records: ARRAY[ANY]
  -- Create an empty book
  make do ... end
  -- Add a name-record pair to the book
  add (name: STRING; record: ANY) do ... end
  -- Return the record associated with a given name
  get (name: STRING): ANY do ... end
end
```

Client

```
1  birthday: DATE; phone_number: STRING
2  b: BOOK; is_wednesday: BOOLEAN
3  create {BOOK} b.make
4  phone_number := "416-677-1010"
5  b.add ("SuYeon", phone_number)
6  create {DATE} birthday.make(1975, 4, 10)
7  b.add ("Yuna", birthday)
8  is_wednesday := b.get("Yuna").get_day_of_week = 4
```

b.get("SuYeon")

ANY

ANY

DATE

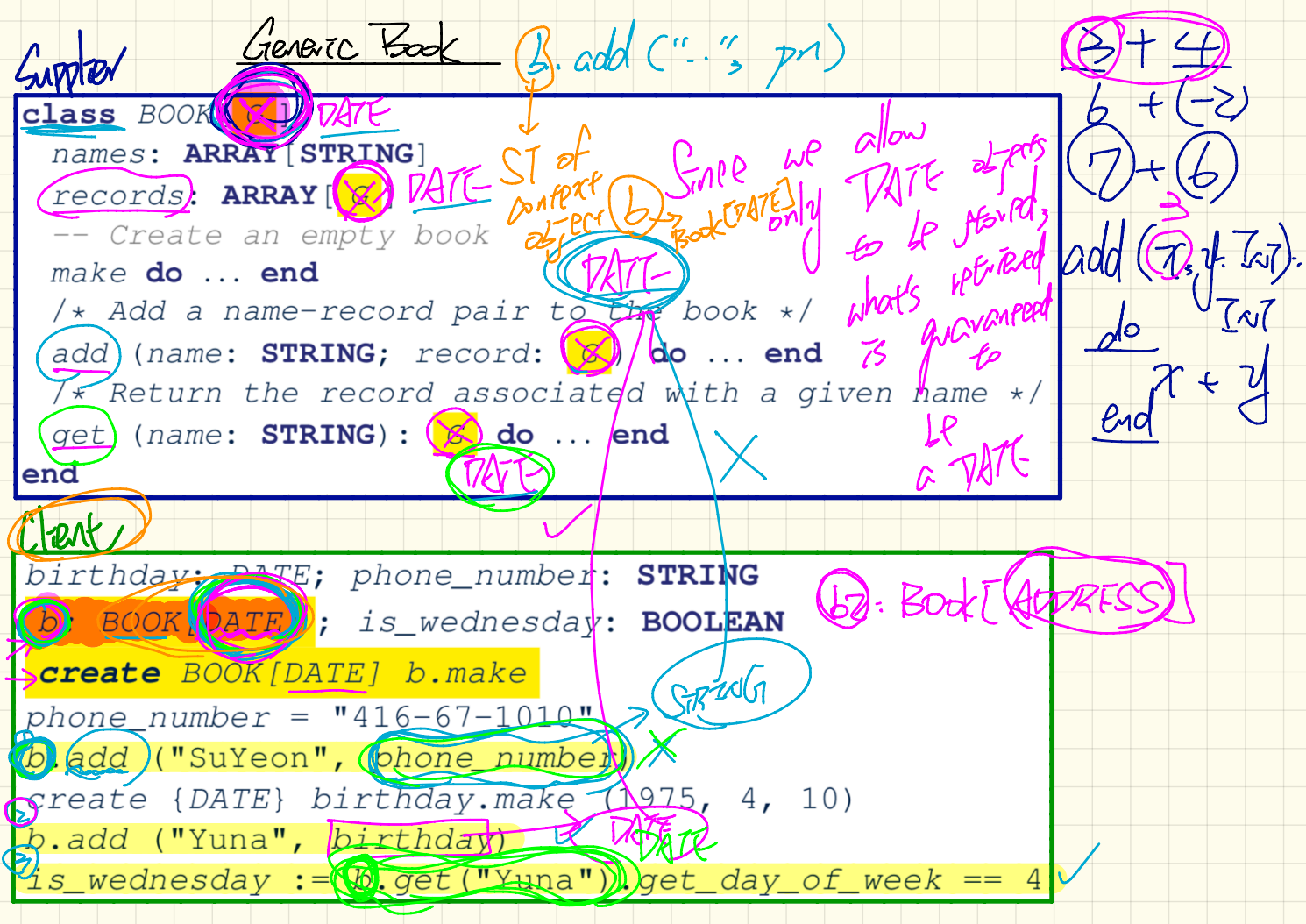
if attached {CI} b.get("Jim") then

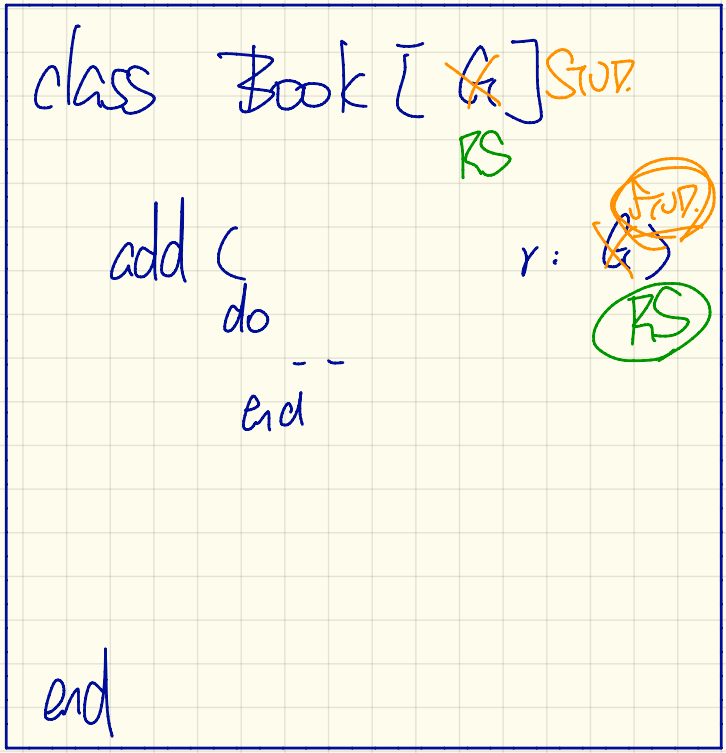
check attached {CI} b.get("Jim") as o then

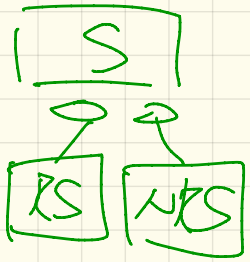
end

X
works
but
violate
SCP

else if .


Supplier
Generic Book
b.add ("...", pn)
class BOOK [X]
names: ARRAY[STRING]
records: ARRAY[X]
-- Create an empty book
make do ... end
/* Add a name-record pair to the book */
add (name: STRING; record: X) do ... end
/* Return the record associated with a given name */
get (name: STRING): X do ... end
end
DATE
DATE
DATE
DATE
ST of compare objects
b
Since we only allow DATE objects to be stored, what's retrieved is guaranteed to be a DATE
X
Client
birthday: DATE; phone_number: STRING
b: BOOK[DATE]; is_wednesday: BOOLEAN
create BOOK[DATE] b.make
phone_number = "416-67-1010"
b.add ("SuYeon", phone_number)
create {DATE} birthday.make (1975, 4, 10)
b.add ("Yuna", birthday)
is_wednesday := b.get("Yuna").get_day_of_week == 4
String
X
DATE
DATE
b2: Book[ADDRESS]
3 + 4
6 + (-2)
7 + 6
add (x, y: INT): INT
do
x + y
end





client:

b1: Book [STUDENT]

b2: Book [RS]

s1: STUDENT

s2: RS

s3: NRS

① b1. add (s1)

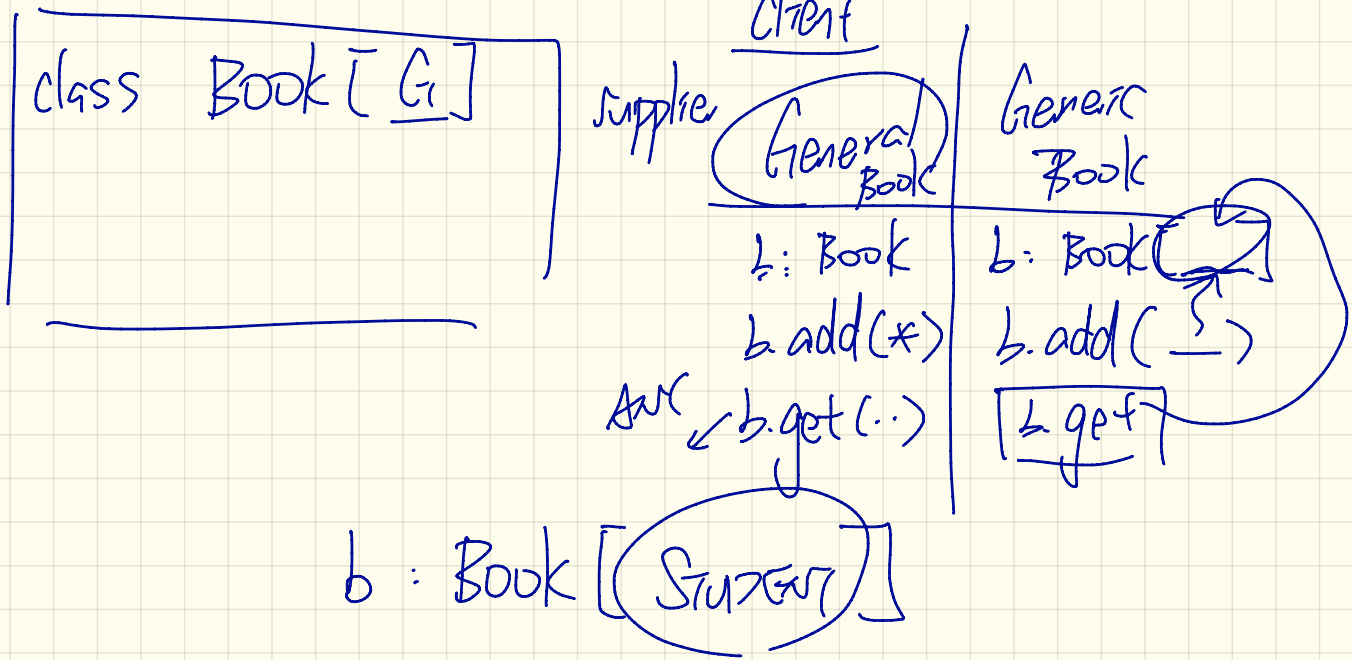
② b1. add (s2)

③ b1. add (s3)

④ b2. add (s1)

⑤ b2. add (s2)

⑥ b2. add (s3)


class Book[G]
Client
Supplier
General Book
Generic Book
b: Book
b.add(*)
b.get(..)
b: Book
b.add(_)
b.get
Anc
b : Book[Student]
b. get(____)
b. add(____)

Instantiating Generic Parameters

Say the **supplier** provides a generic `DICTIONARY` class:

```
class DICTIONARY[V, K] -- V type of values; K type of keys
  add_entry (v: V; k: K) do ... end
  remove_entry (k: K) do ... end
end
```

Clients use `DICTIONARY` with different degrees of instantiations:

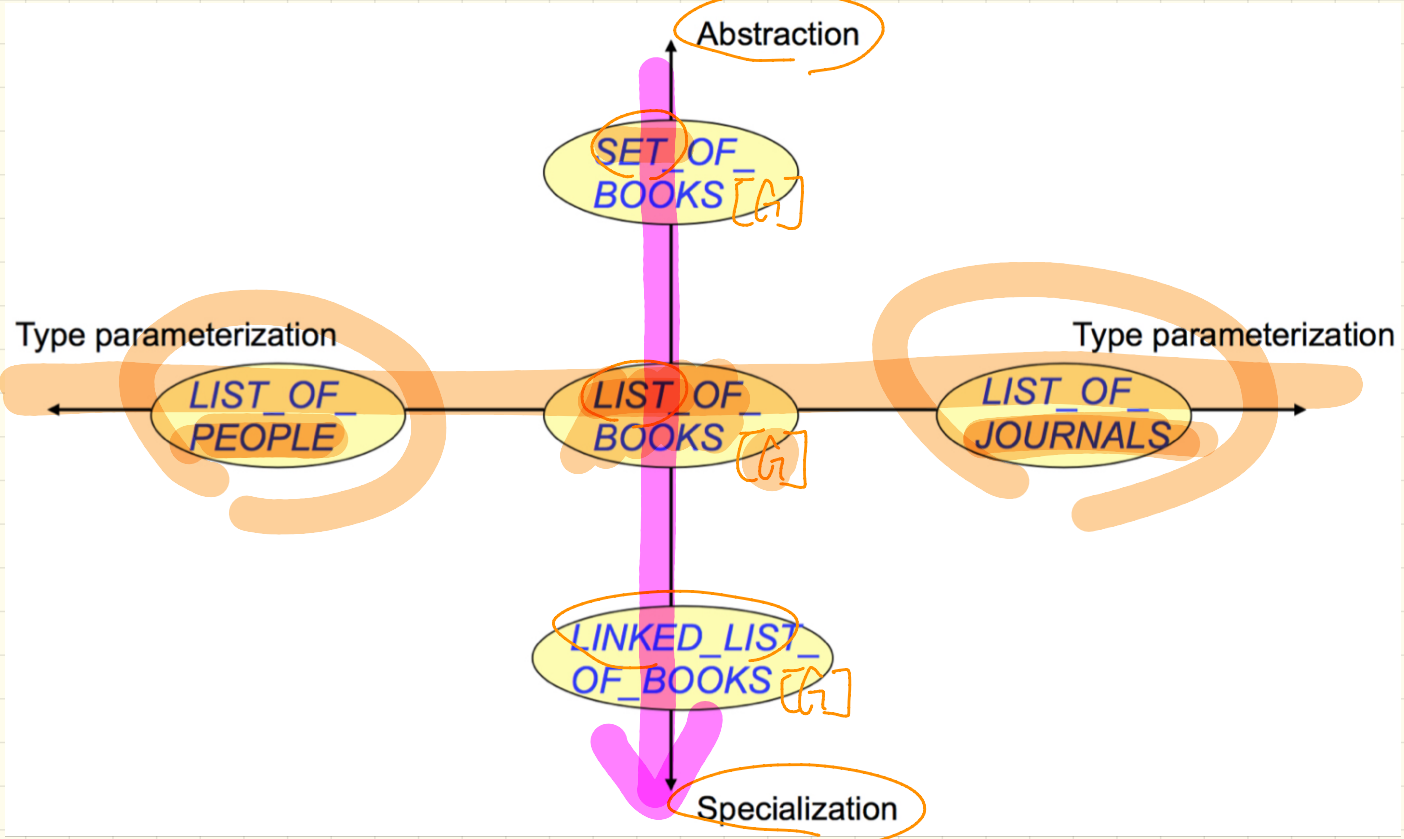
```
class DATABASE_TABLE[K, V]
  imp: DICTIONARY[V, K]
end
```

e.g., Declaring `DATABSE_TABLE[INTEGER, STRING]` instantiates `DICTIONARY[STRING, INTEGER]`.

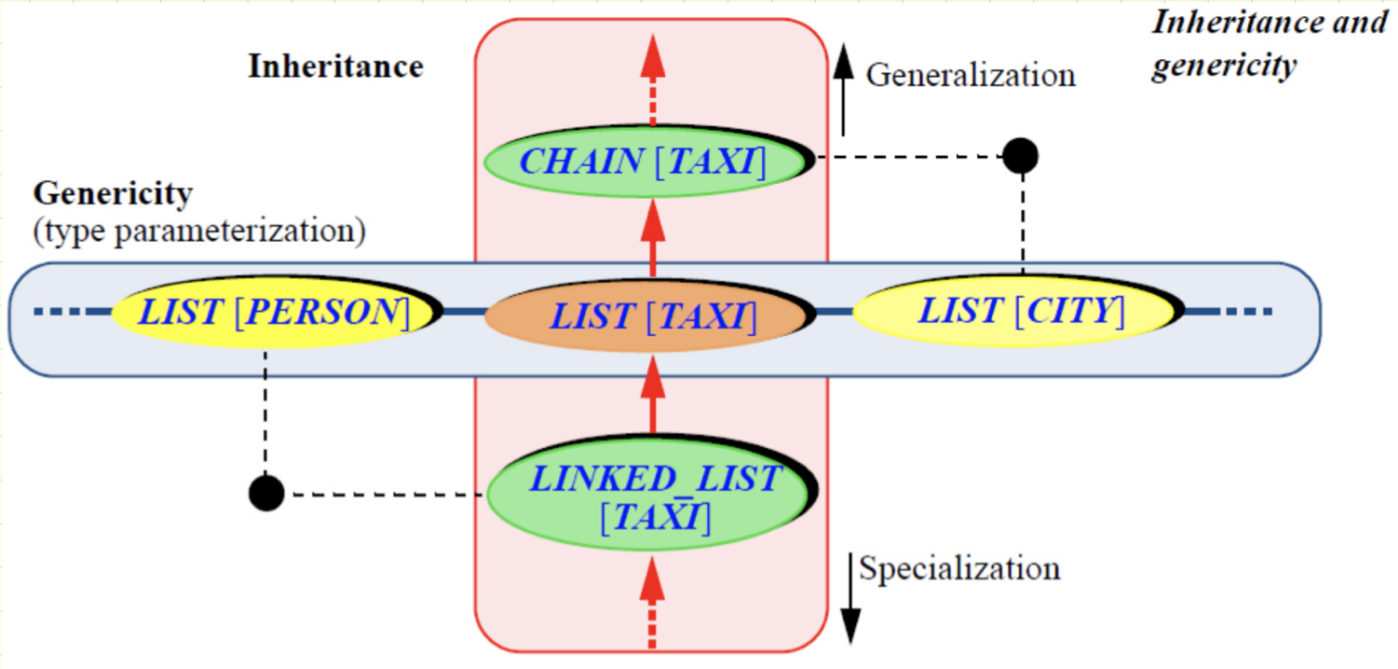
```
class STUDENT_BOOK[V]
  imp: DICTIONARY[V, STRING]
end
```

e.g., Declaring `STUDENT_BOOK[ARRAY[COURSE]]` instantiates `DICTIONARY[ARRAY[COURSE], STRING]`.

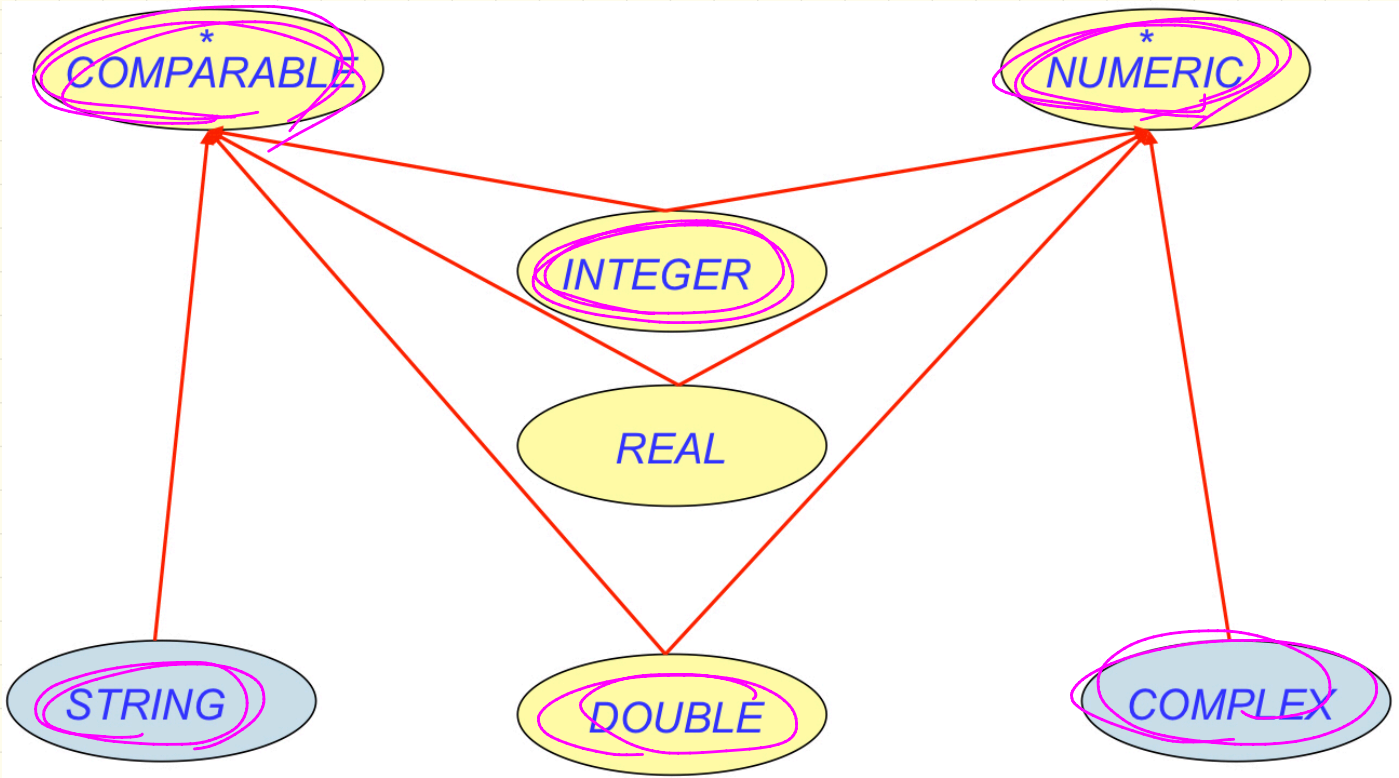
Generics vs. Inheritance (1)

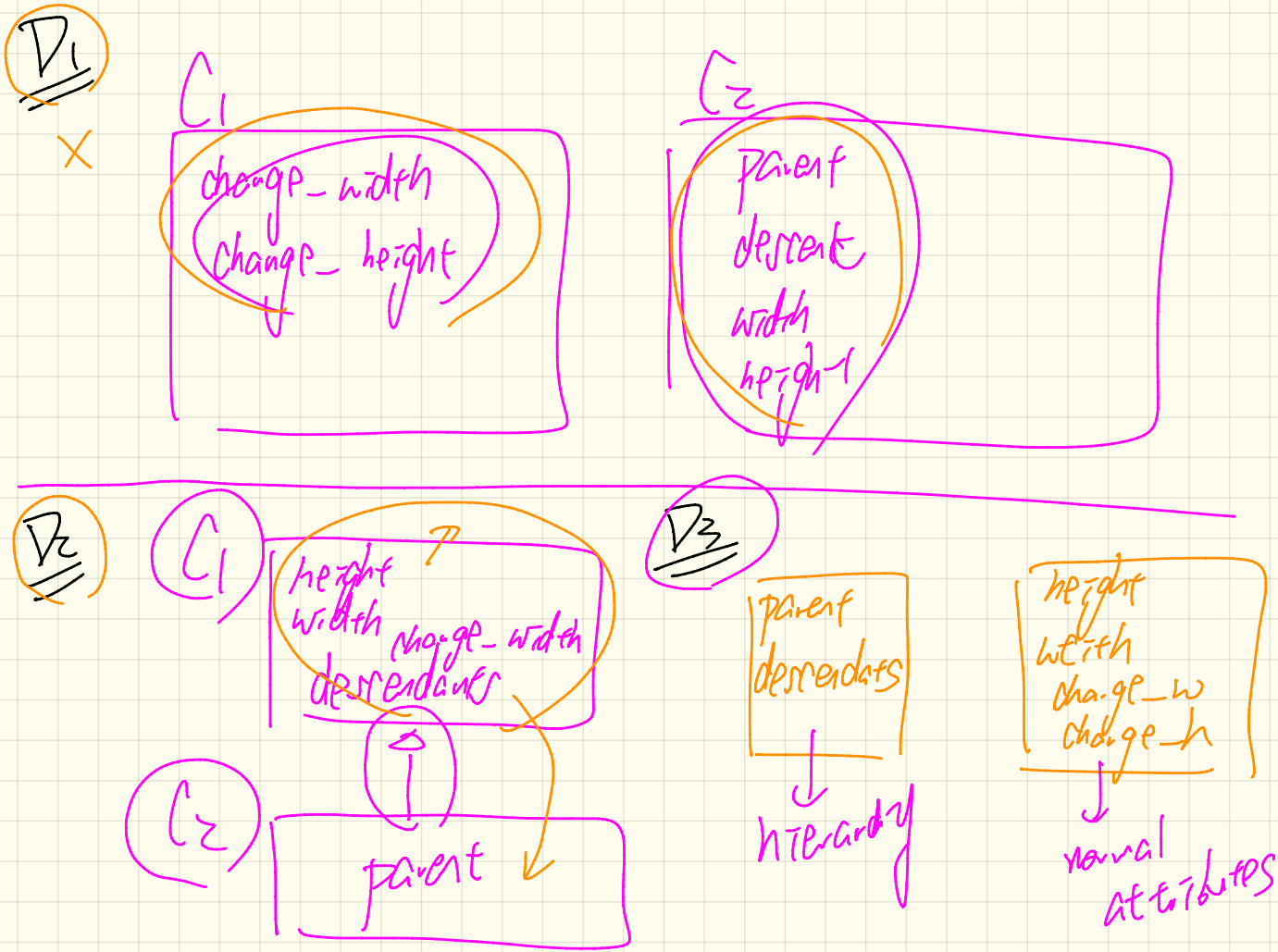


Generics vs. Inheritance (2)



Multiple Inheritance : Example

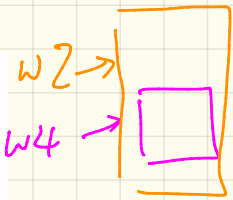



D1
X
C1
change_width
change_height
C2
parent
descent
width
height
D2
C1
height
width
change_width
descendants
C2
parent
D3
parent
descendants
hierarchy
height
width
cha.ge_w
change_h
normal
attributes

Multiple Inheritance : Exercise

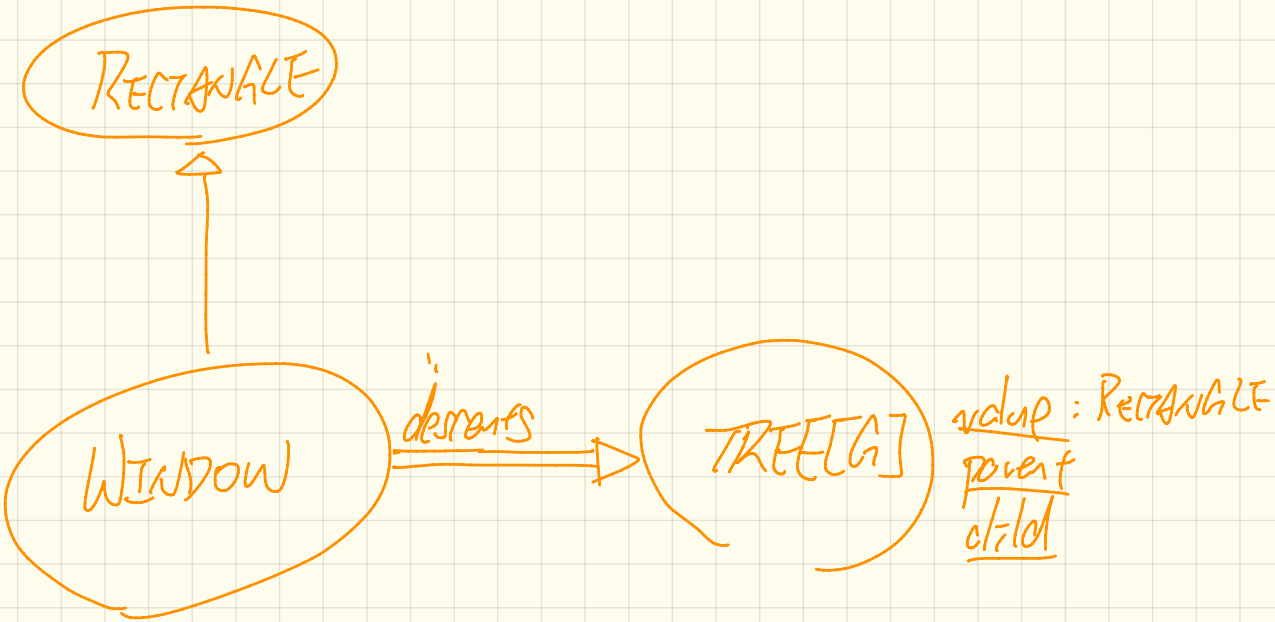
```
class RECTANGLE
  feature -- Queries
    width, height: REAL
    xpos, ypos: REAL
  feature -- Commands
    make (w, h: REAL)
    change_width
    change_height
    move
end
```

```
class TREE[G]
  feature -- Queries
    parent: TREE[G]
    descendants: LIST[TREE[G]]
  feature -- Commands
    add_child (c: TREE[G])
end
```

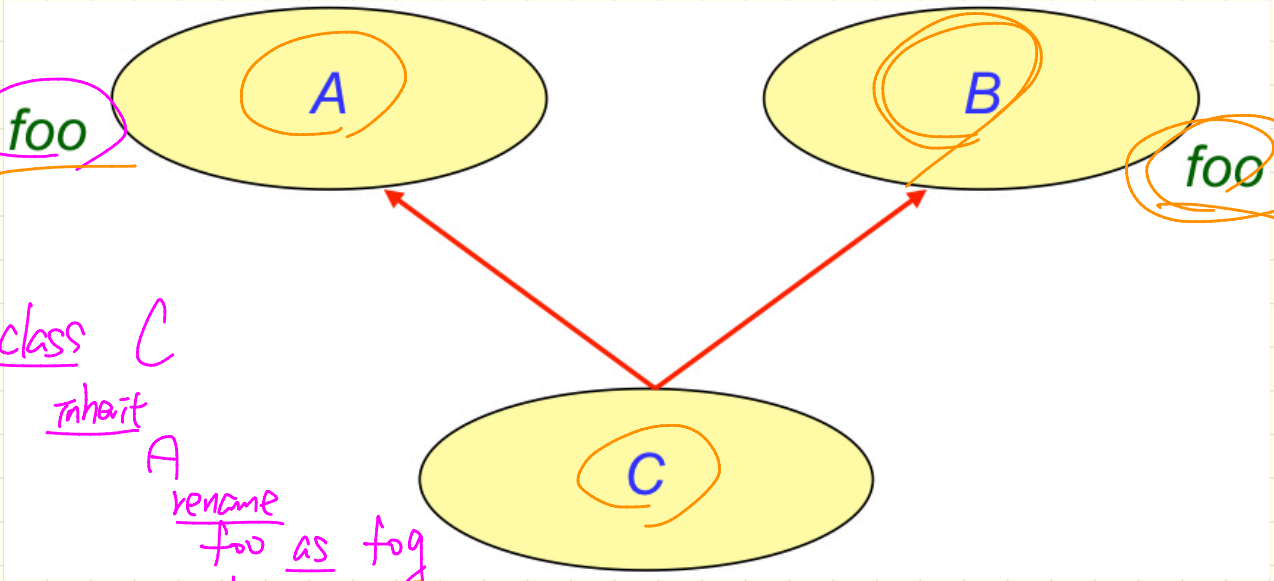



```
class WINDOW
  inherit
    RECTANGLE
    TREE[WINDOW]
feature
  add (w: WINDOW)
end
```

```
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
```


RECTANGLE
WINDOW
desens
TREE[G]
value : RECTANGLE
parent
child

Multiple Inheritance : Name Clashes




```
class C
  inherit
    A
      rename
        foo as fog
      end
    B
      rename
        foo as zoo
      end
```