Lecture 13
Thursday October 24

General Book

① compilation? ✓ ② runtime ✓ violation.

Supplier

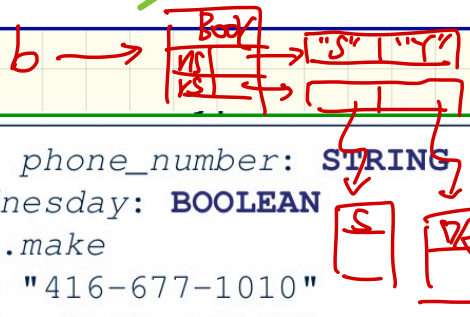
```
class BOOK
  names: ARRAY[STRING]
  records: ARRAY[ANY]
  -- Create an empty book
  make do ... end
  -- Add a name-record pair to the book
  add (name: STRING; record: ANY) do ... end
  -- Return the record associated with a given name
  get (name: STRING): ANY do ... end
end
```

```
check attached {DATE} b.get("SuYeon") as d then
  is_w := d.get_day_of_week = 4
end
```

ANY

DATE

r.get_day_of_week

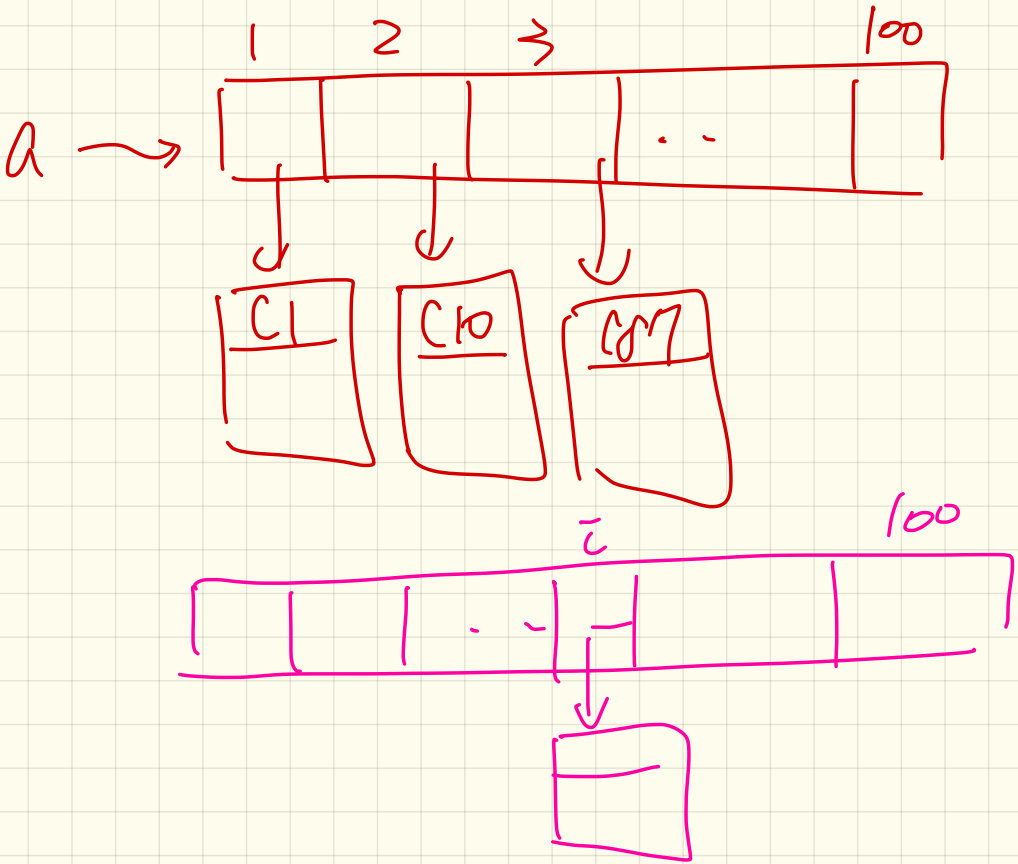


Client

```
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(19?, 4, 10)
7  b.add ("Yuna", birthday)
8  is_wednesday := b.get("Yuna").get_day_of_week = 4
```

ANY

X

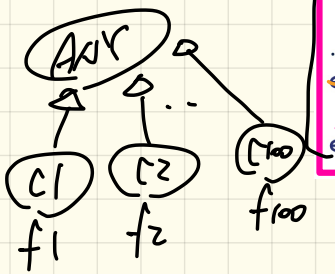

1
2
3
100
a
C1
C10
C97
i
100

General Book violates Single Choice Principle

Storage

```
rec1: C1
... -- declarations of rec2 to rec99
rec100: C100
create {C1} rec1.make(...) ; b.add(..., rec1)
... -- additions of rec2 to rec99
create {C100} rec100.make(...) ; b.add(..., rec100)
```

Retrievals




```
-- assumption: 'f1' specific to C1, 'f2' specific to C2, etc.
if attached {C1} b.get("Jim") as c1 then
  c1.got("Jim").f1
... -- cases for C2 to C99
elseif attached {C100} b.get("Jim") as c100 then
  c100.f100
end
```

repeat for C1

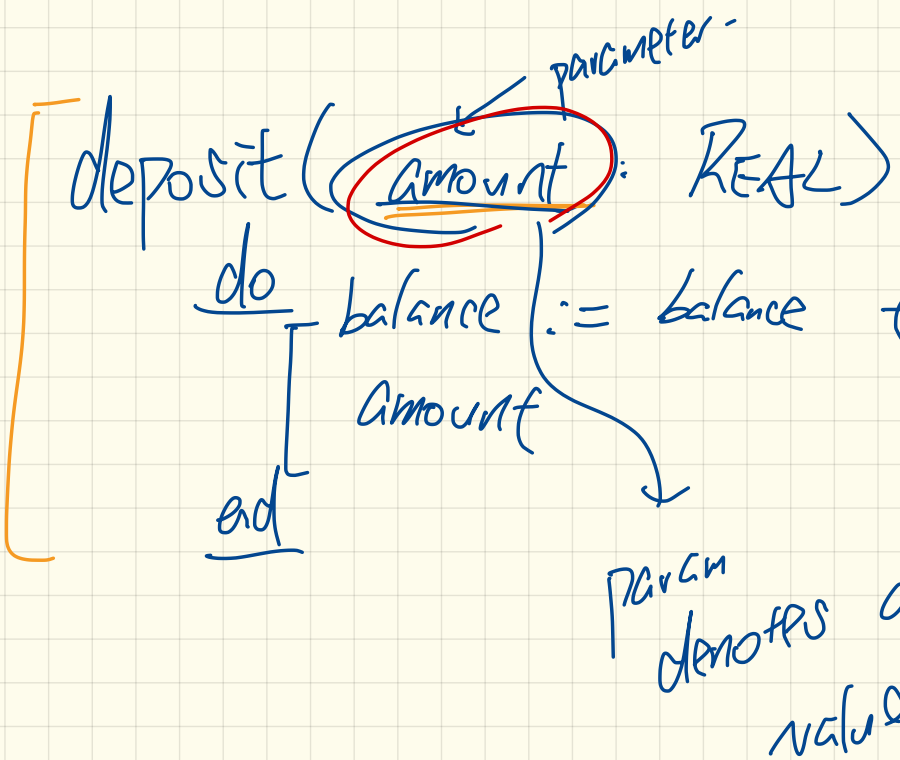
else: attached {C101} -- then -- repeat for C2?

```
-- assumption: 'f1' specific to C1, 'f2' specific to C2, etc.
if attached {C1} b.get("Jim") as c1 then
  c1.got("Jim").f1
... -- cases for C2 to C99
elseif attached {C100} b.get("Jim") as c100 then
  c100.f100
end
```

else: attached {C101} -- then --

What if a new type C101 is introduced?

What if type C100 becomes obselete?



acc.deposit (23.4)

deposit (46.8)

Generic Book

Supplier

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

generic parameter denoting some type, not value

DATE

DATE

use of parameter G.

DATE

DATE

local
acc: Account
b: Book[acc]

Client

type that instantiates G of Book.

```
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
```

① ② STRING.
③
④ DATE
⑤

b2: Book[STUDENT]

s1, s2: STUDENT

create {RS} s1.make(..)

create {NRS} s2.make(..)

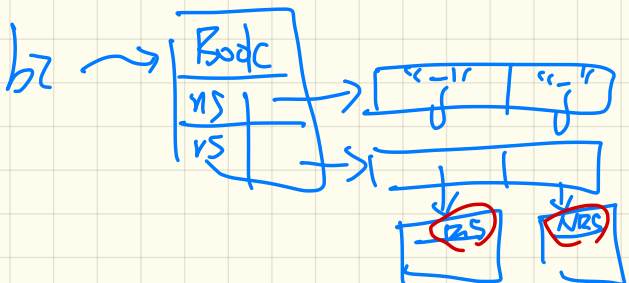
```
class BOOK[G]
  names: ARRAY[STRING]
  records: ARRAY[G]
  -- Create an empty book
  make do ... end
  /* Add a name-record pair to the book */
  add (name: STRING; record: G) do ... end
  /* Return the record associated with a given name */
  get (name: STRING): G do ... end
end
```

STUDENT

STU.

STU.

STU.

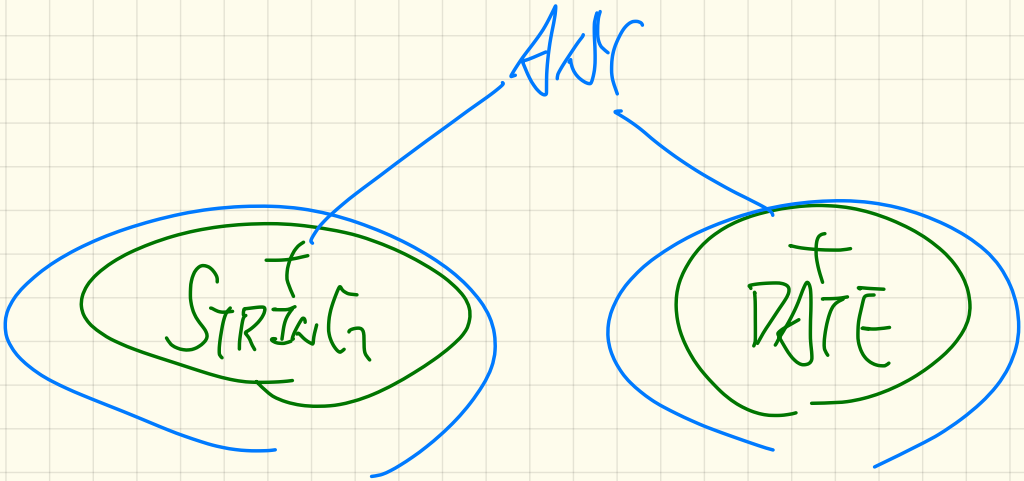


b2.add("jim", s1)

b2.add("jenny", s2)

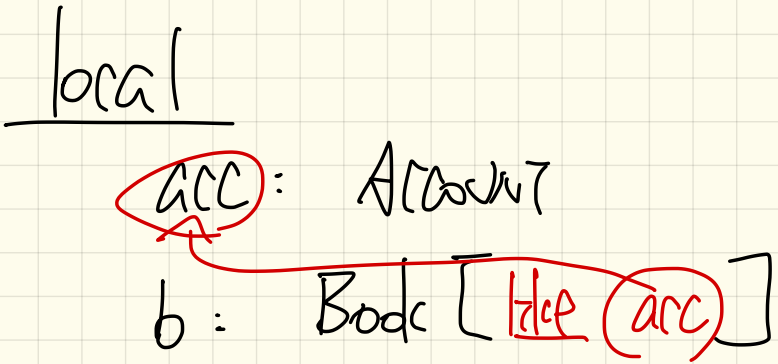
s1 s

b3 : Book[ANY]



class Book2 [like acc] X

class Book [G -> Student]


local
acc : Account
b : Book [title acc]

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
```

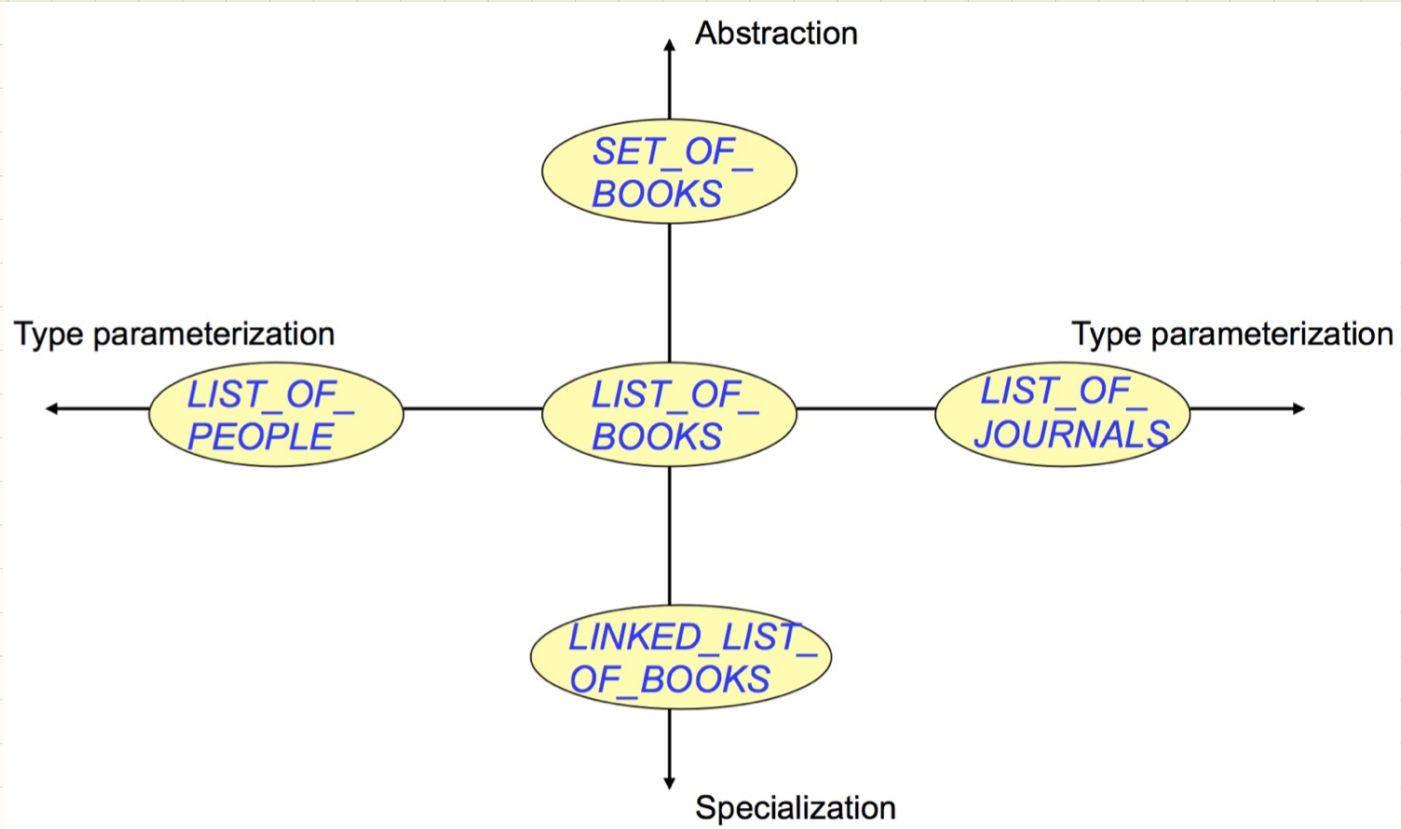
DATABASE_TAB[I , S]

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

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

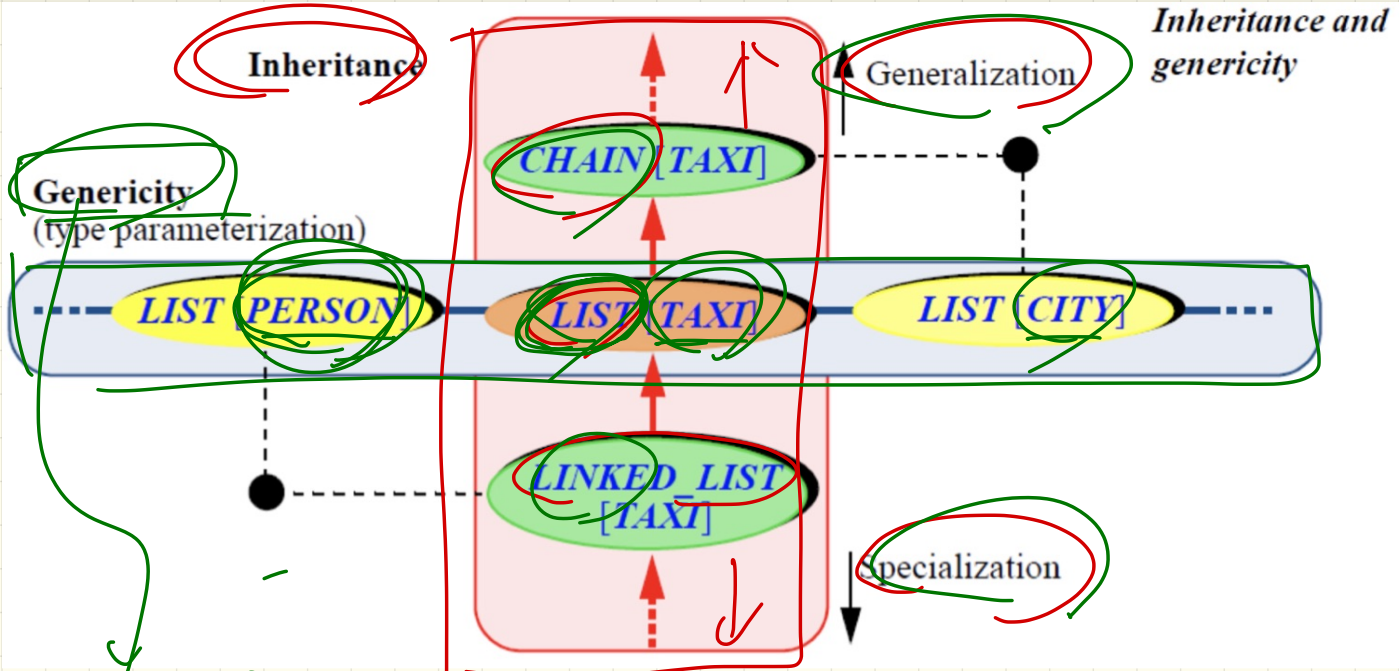
e.g., Declaring `STUDENT_BOOK[ARRAY[COURSE]]` instantiates `DICTIONARY[ARRAY[COURSE], STRING]`.

Generics vs. Inheritance (1)

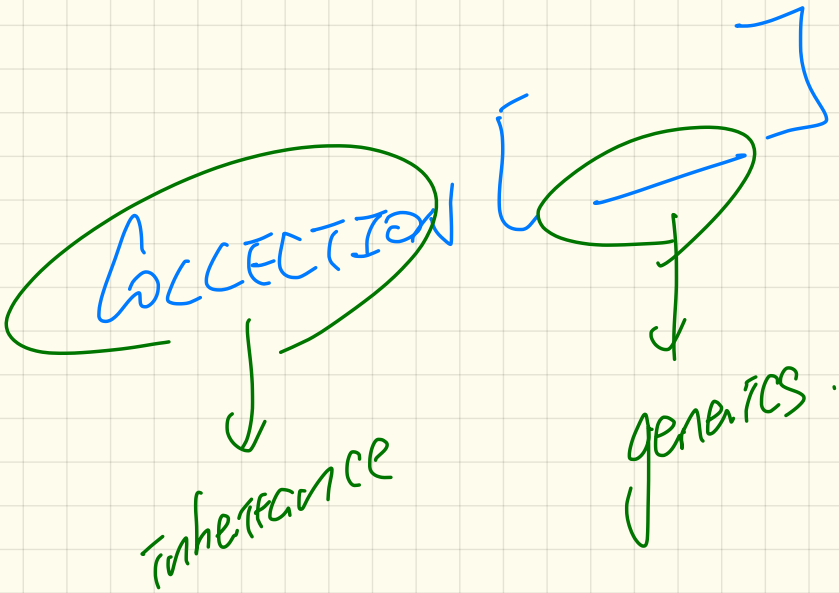


Generics vs. Inheritance (2)

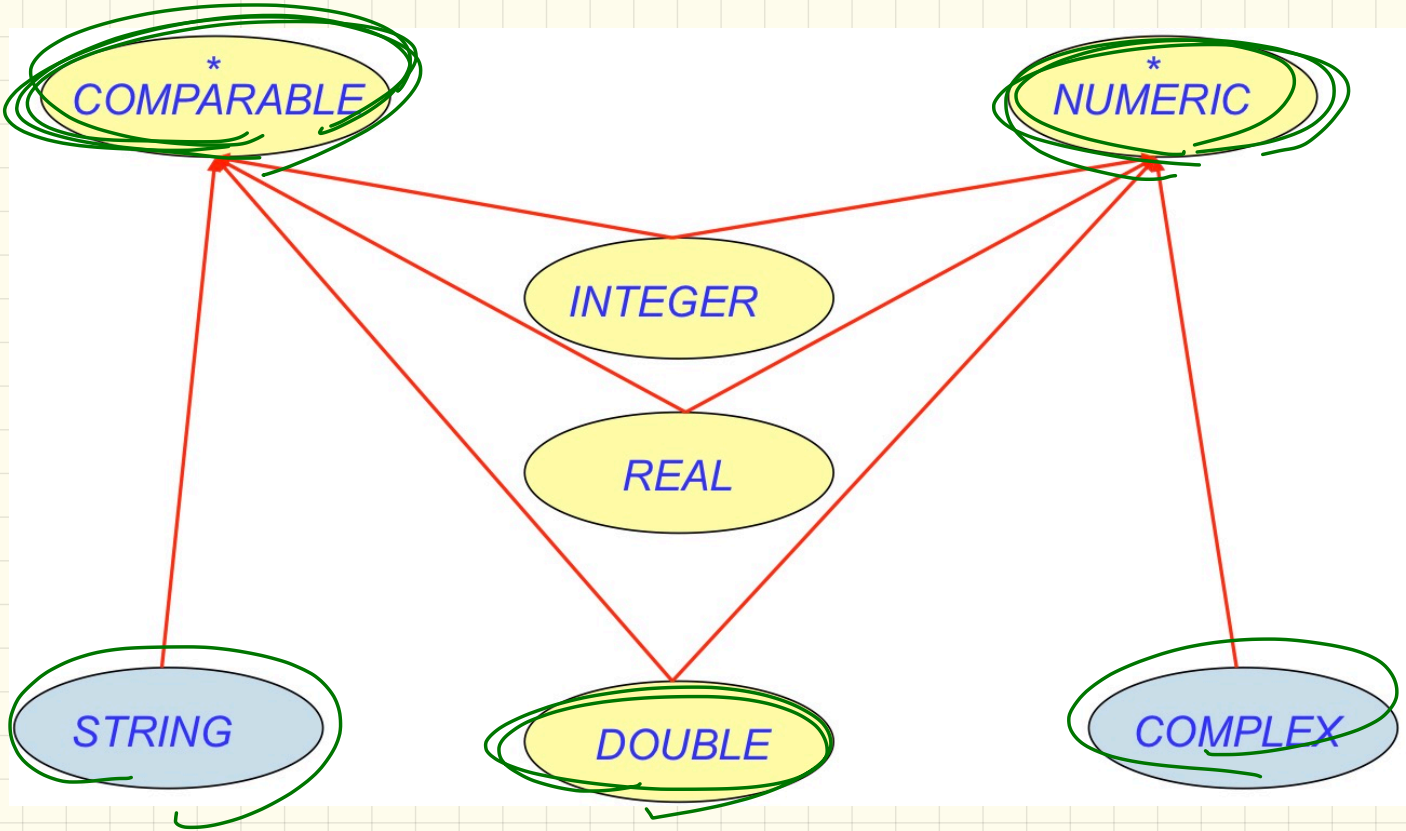
class LIST[G]

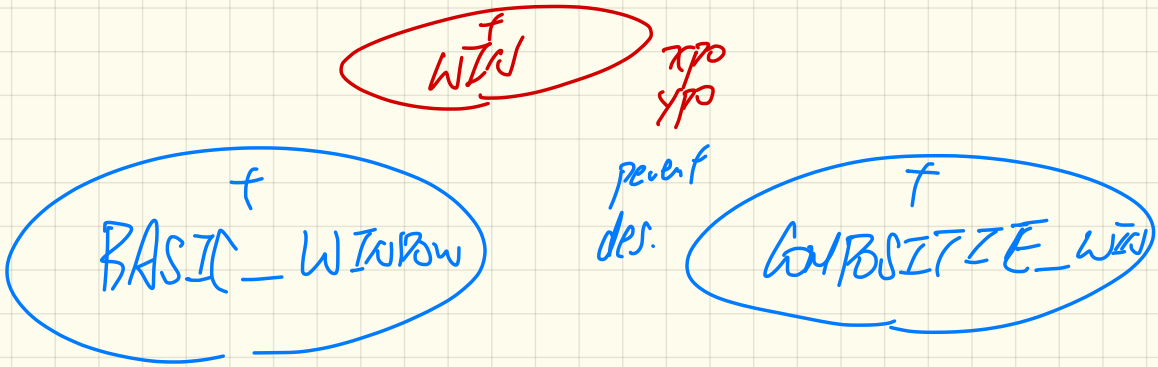


parameterize the type of members in a collection


COLLECTION
inheritance
generics.

Multiple Inheritance: Example




f
WIN
xpos
ypos
f
BASIC_WINDOW
peerf
des.
f
GUIRESIZE_WIN

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
```

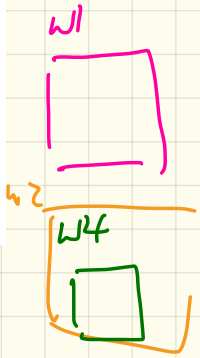
TREE[G → WINDOW]

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

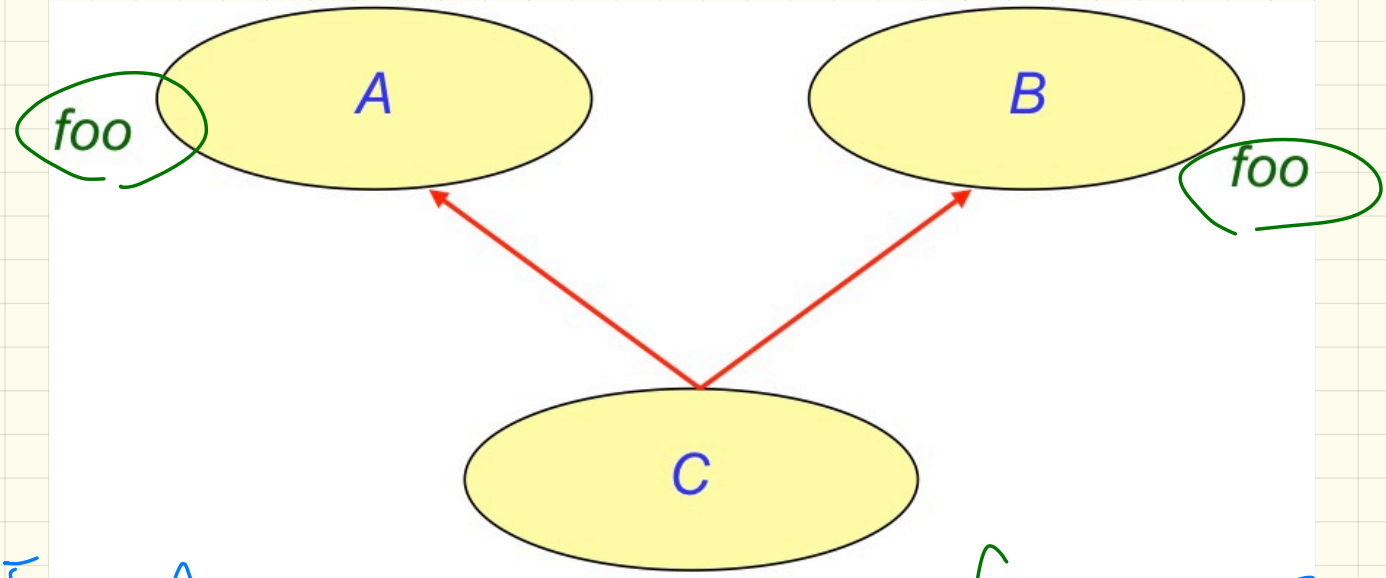
orthogonal

```
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 rename foo as fog
            B
```

C: C
C.foo → B
C.fog → foo from A

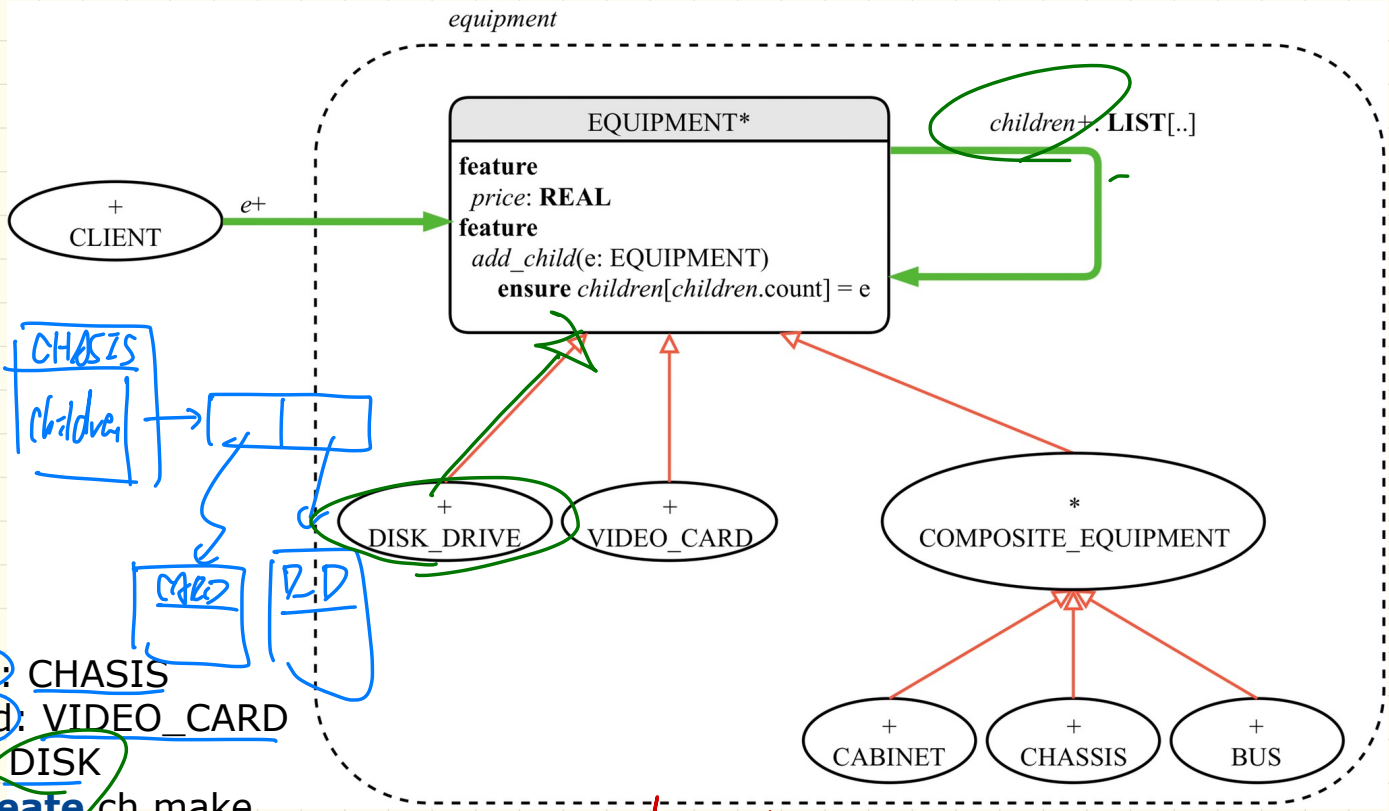
Overloading

Java

```
deposit (String id, int amount)
deposit (int amount)
```

Eiffel

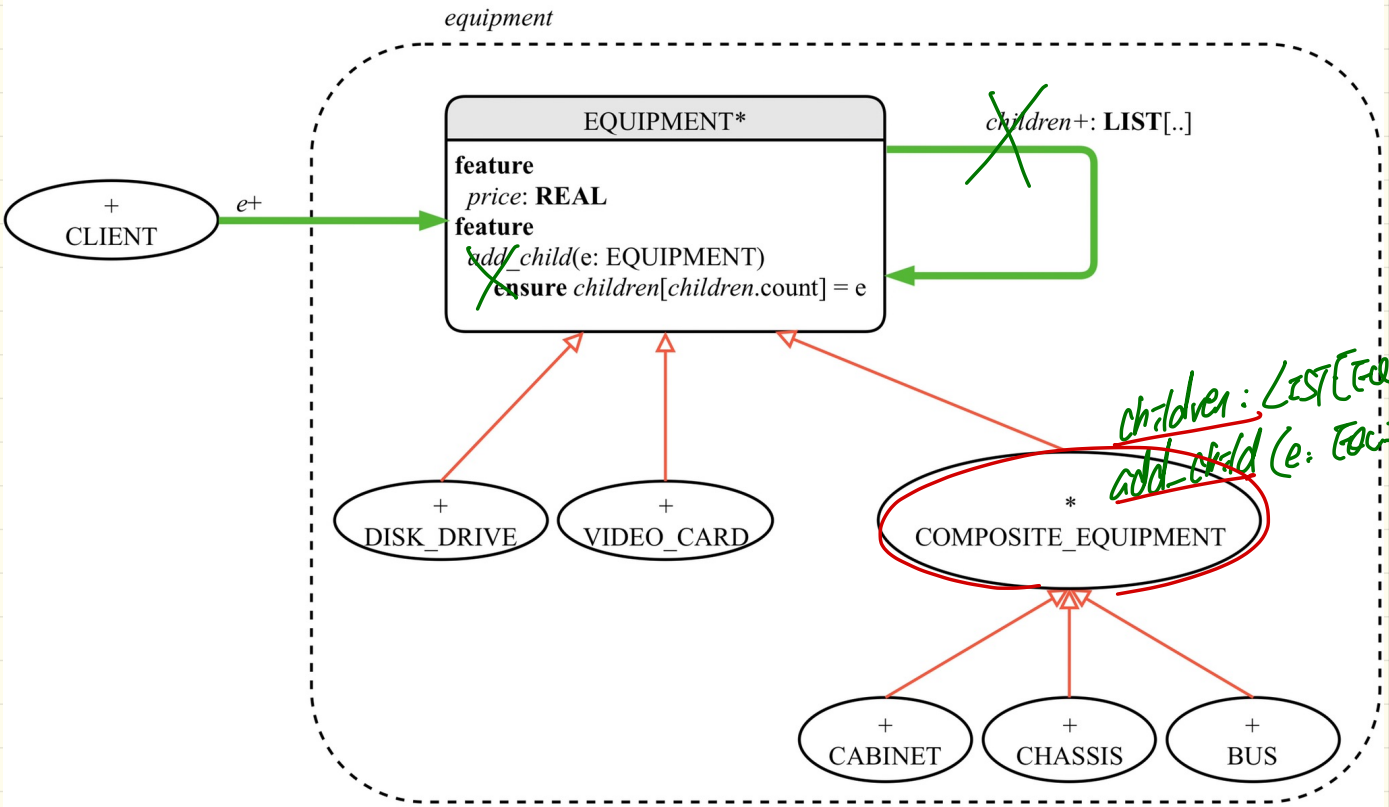
```
deposit_1 (_____, _____)
deposit_2 (_____)
```

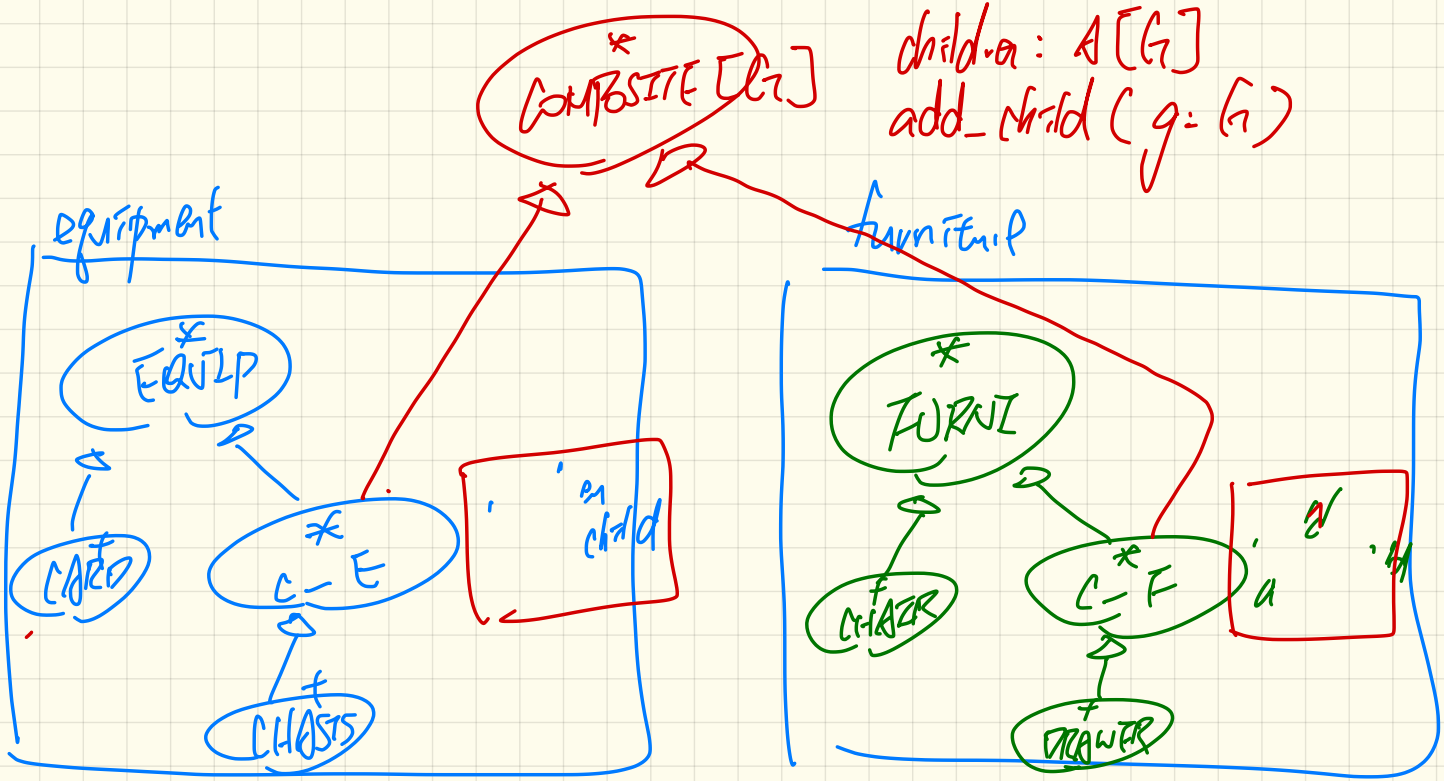


ch: CHASIS
crd: VIDEO_CARD
d: DISK
create ch.make
create crd.make
create d.make
ch.add_child(crd)
ch.add_child(d)

d.add_child(crd)
SI: DISK
defined in EQUIP.

First Design Attempt


equipment
EQUIPMENT*
feature
price: REAL
feature
add_child(e: EQUIPMENT)
ensure children[children.count] = e
children+: LIST[..]
+
CLIENT
e+
+
DISK_DRIVE
+
VIDEO_CARD
*
COMPOSITE_EQUIPMENT
+
CABINET
+
CHASSIS
+
BUS


COMPOSITE[G]
child_a : A[G]
add_child(g: G)
equipment
furniture
EQUIP
CARD
C_E
CHASSIS
child
FURNI
CHAIR
C_F
DRAWER
a