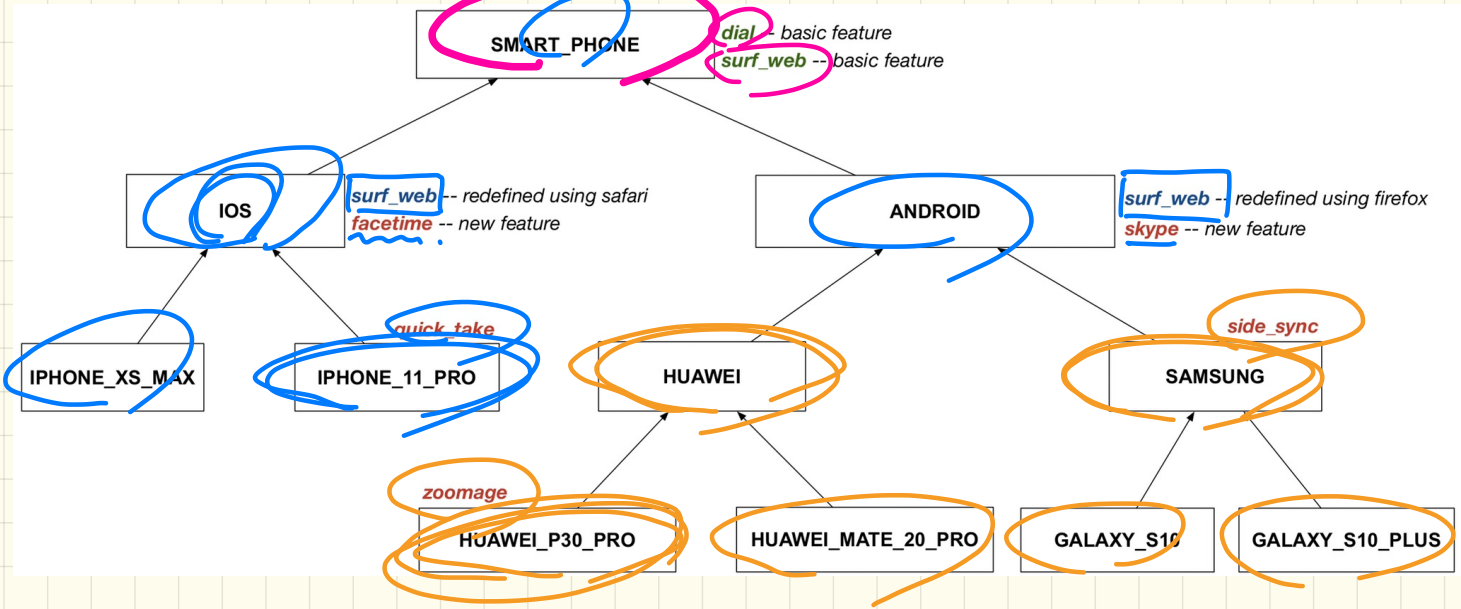
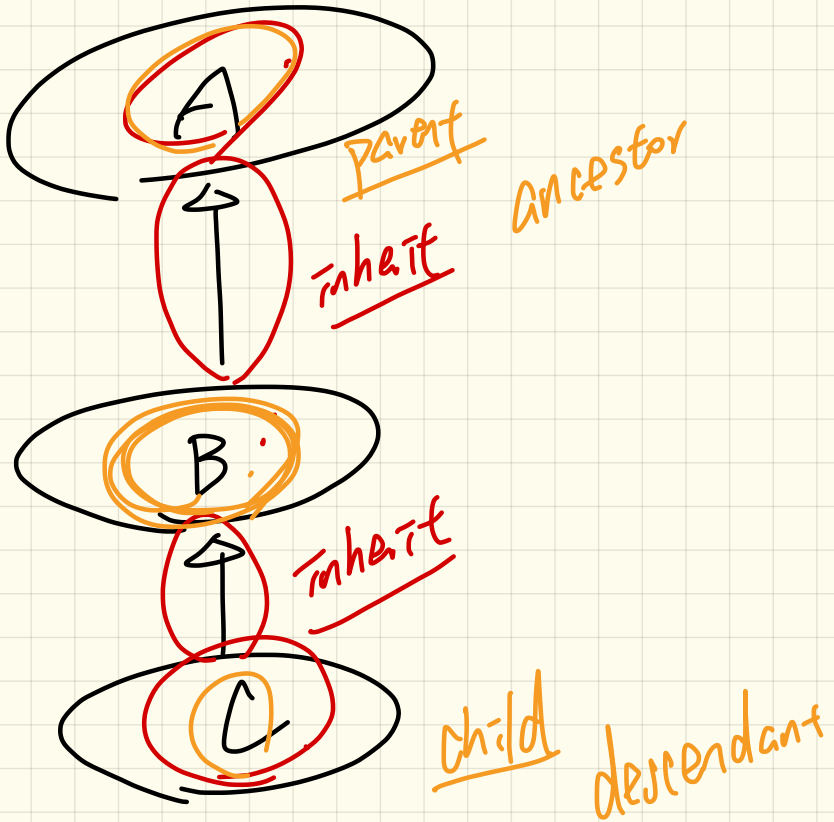
Lecture 16
Wednesday March 4

Multi-Level Inheritance Hierarchy of Smartphones



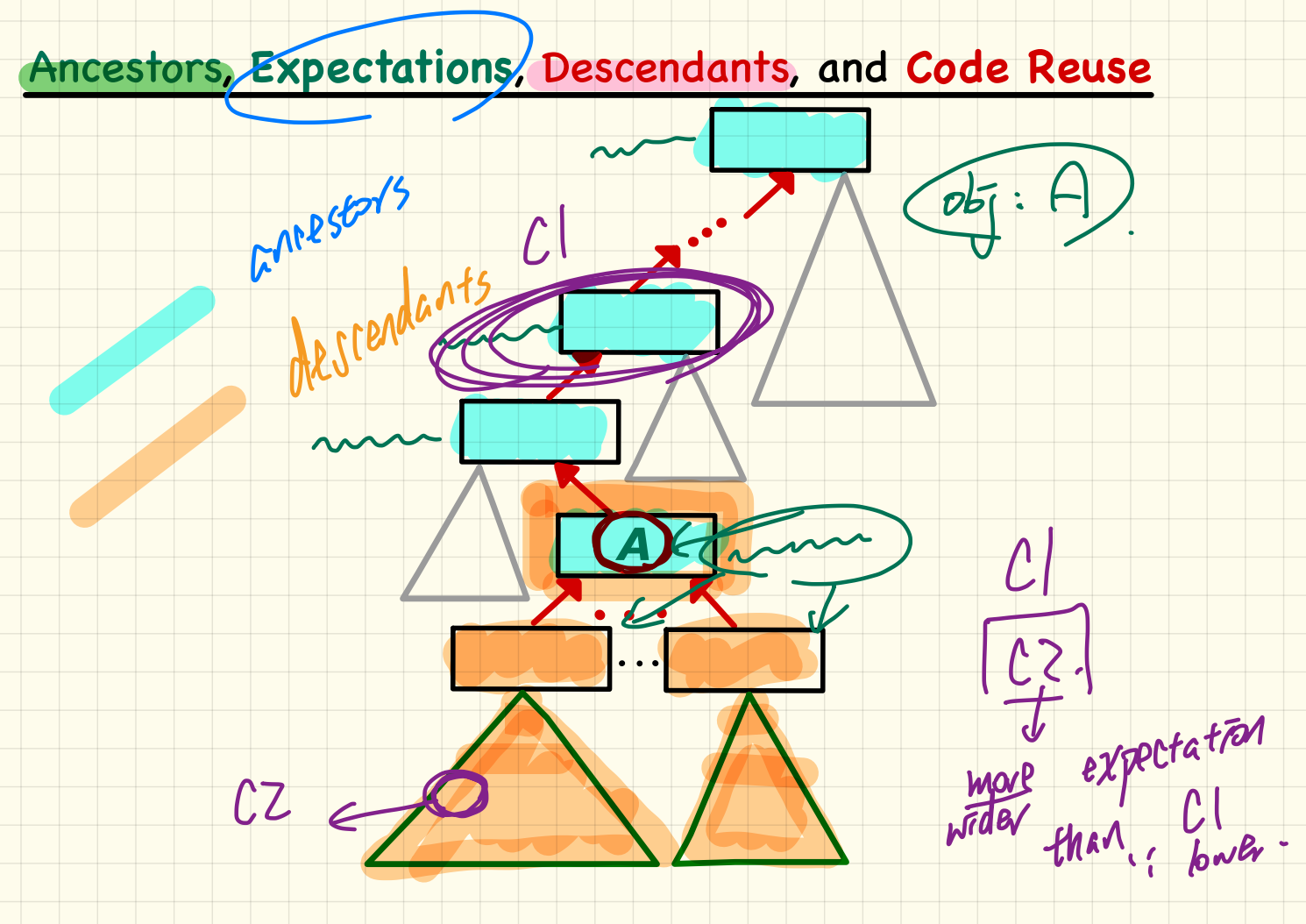


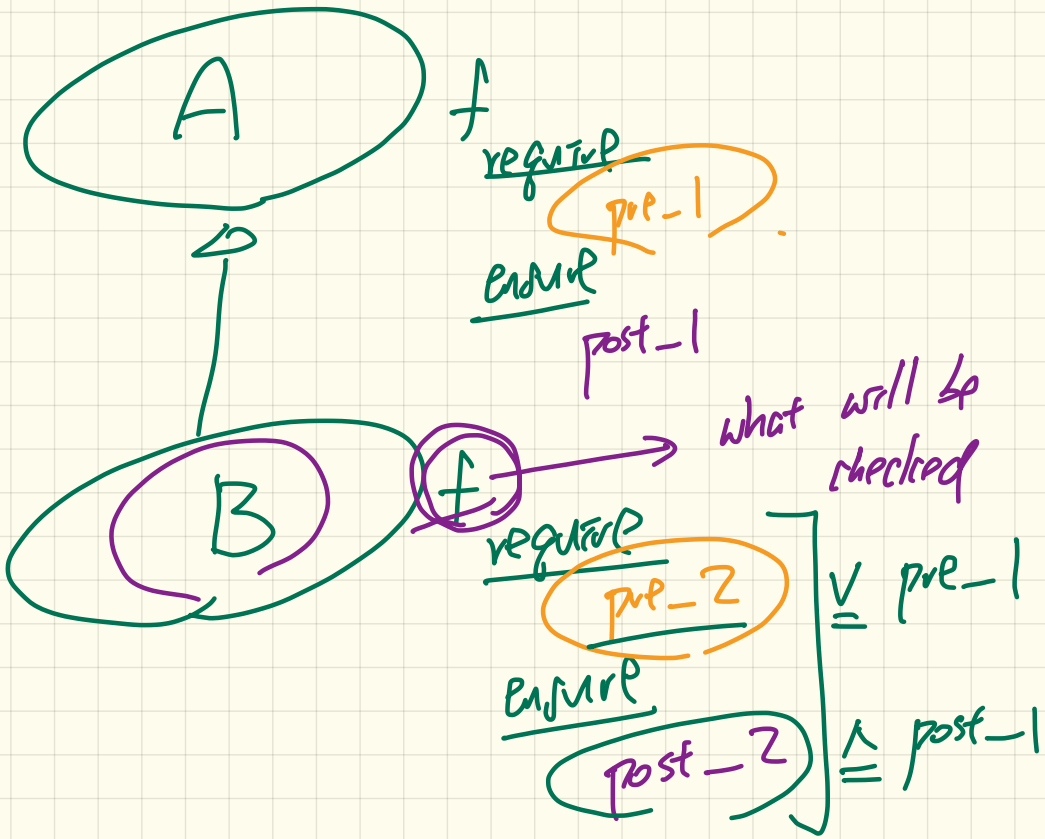
C < B

B < A

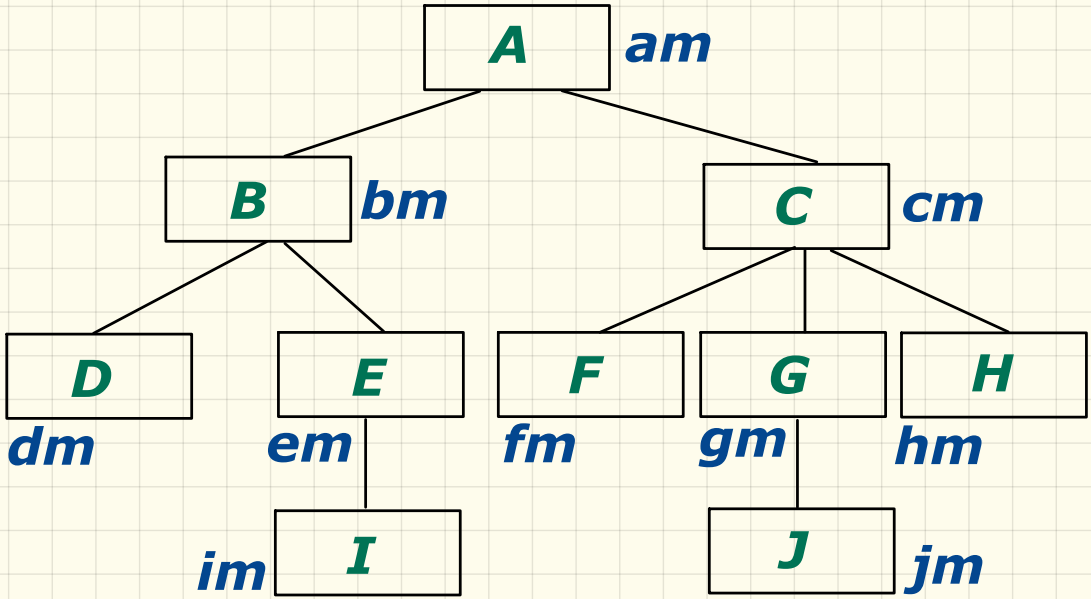
C < A

Ancestors, Expectations, Descendants, and Code Reuse



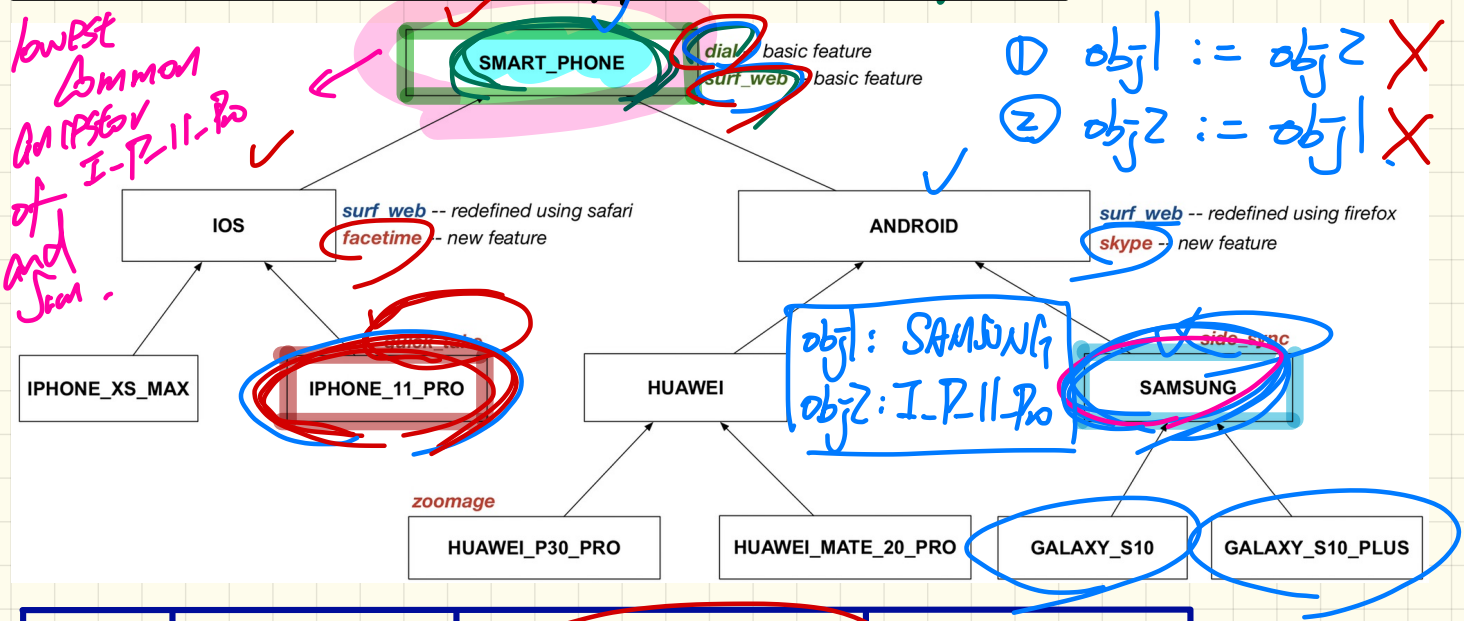

A
f
require
pre_1
ensure
post_1
B
f
what will be checked
require
pre_2
∨ pre_1
ensure
post_2
∧ post_1

Inheritance Forms a Type Hierarchy (1)

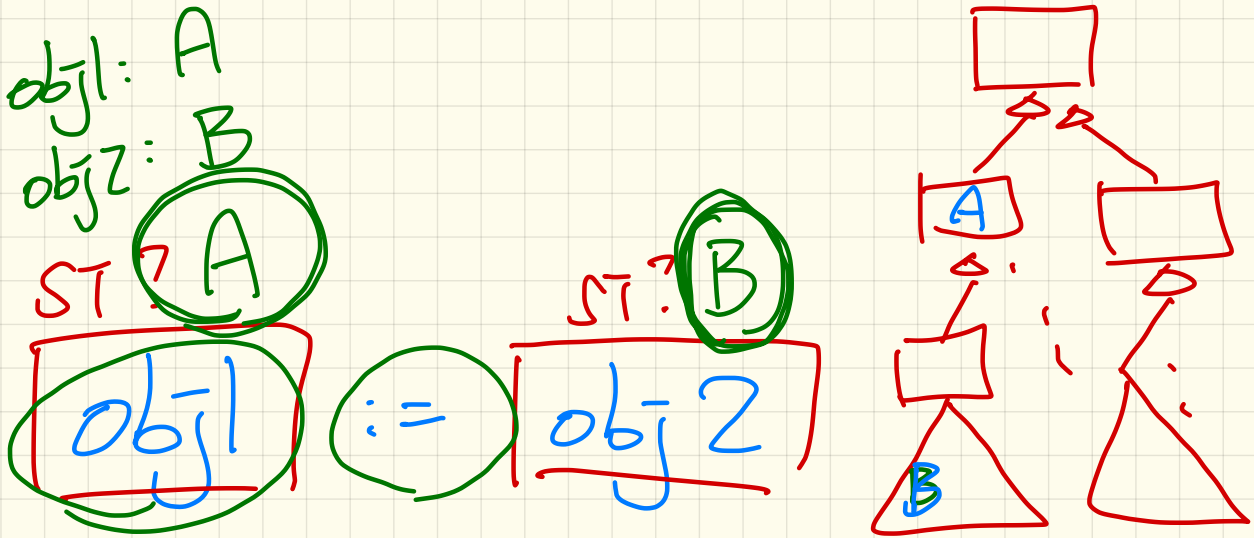


	ancestors	expectations	descendants
B			
G			
J			

Inheritance Forms a Type Hierarchy (2)

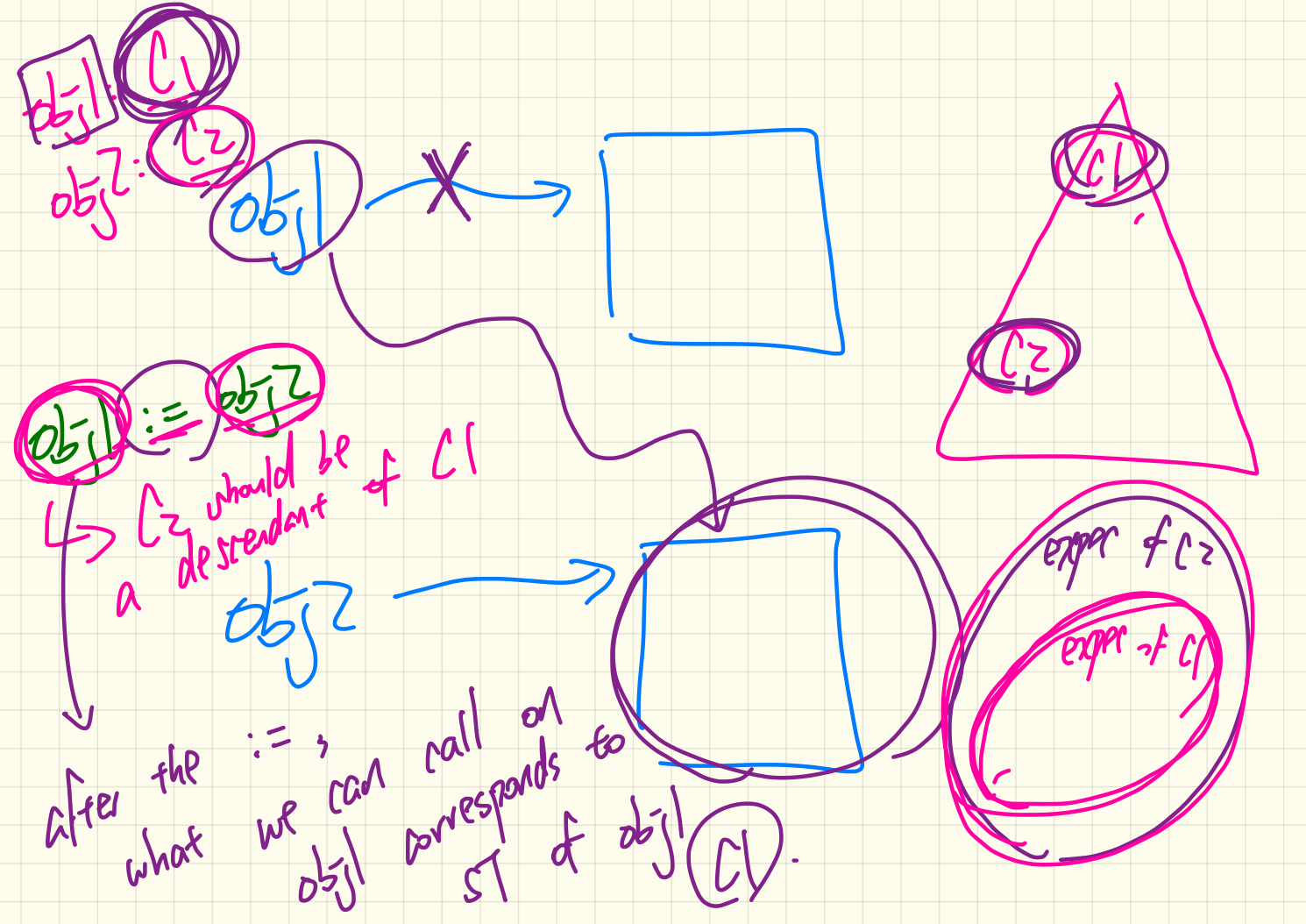


	ancestors	expectations	descendants
S_P	S_P	dial, surf_web	all classes in hierarchy
S, A, S_P	S, A, S_P	dial, surf_web, skype, side_sync	3 classes
I_11_P, IOS, S_P	I_11_P, IOS, S_P	dial, surf_web, face, g_t	

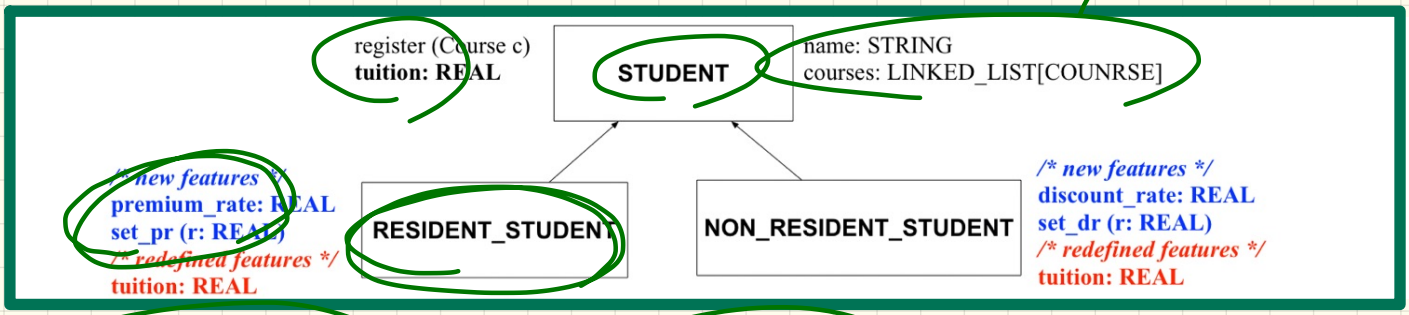


Q: Compile?

expectation of B should be at least as wide/many as the exper. of A.


obj1 : C1
obj2 : C2
obj1
obj1 := obj2
C2 should be a descendant of C1
obj2
after the :=, what we can call on obj1 corresponds to ST of obj1 C1.
C1
C2
expr of C2
expr of C1

Reference Variables: Static Type



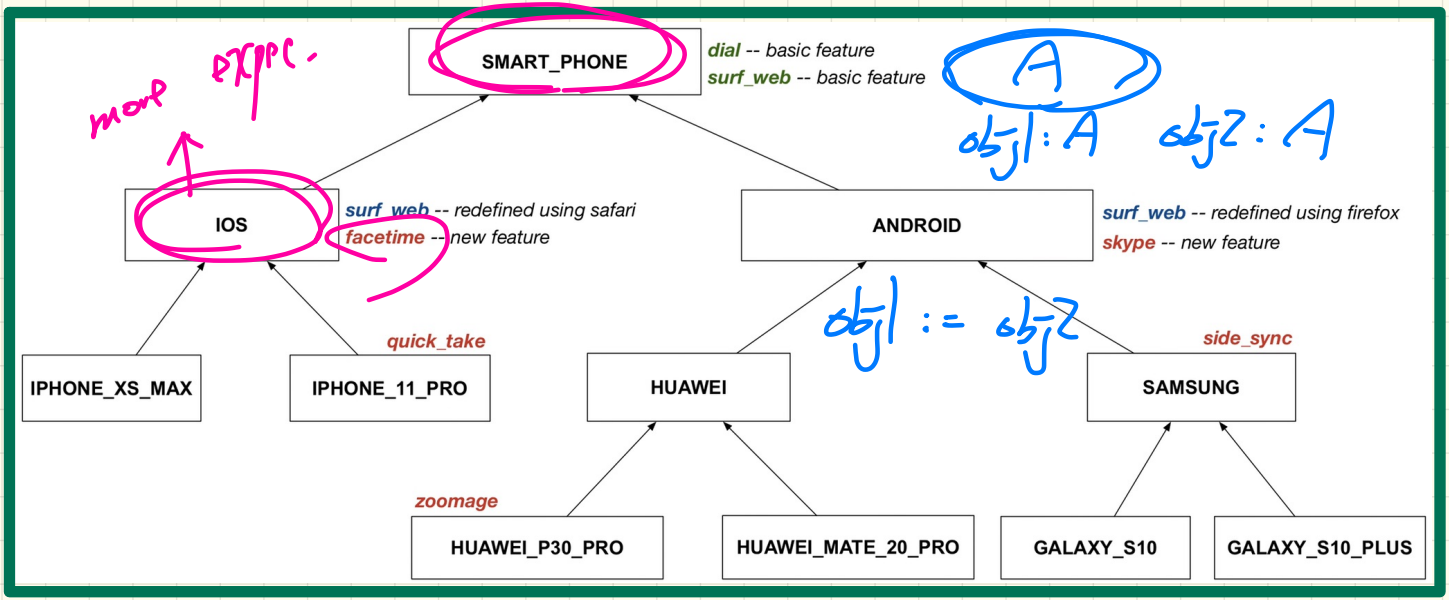
Design 1:

jim: **STUDENT**

Design 2:

jim: **RESIDENT_STUDENT**

Reference Variables: Static Type



Design 1:

mp: SMART_PHONE

mp.facetime X

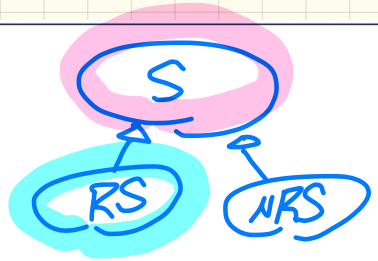
Design 2:

mp: IOS

mp.facetime ✓

Change of Dynamic Type

```
1   test_polymorphism_students
2    local
3      jim: STUDENT
4      rs: RESIDENT_STUDENT
5      nrs: NON_RESIDENT_STUDENT
6    do
7      create {STUDENT} jim.make ("J. Davis")
8      create {RESIDENT_STUDENT} rs.make ("J. Davis")
9      create {NON_RESIDENT_STUDENT} nrs.make ("J. Davis")
10     jim := rs   ✓
11     rs := jim   ×
12     jim := nrs  ✓
13     rs := jim   ×
14   end
```



STUDENT	
n.	
cs.	

RESIDENT_S.	
n.	
cs.	
pr.	

NON_RESI_S.	
n.	
cs.	
dr.	

Testing of Dynamic Binding

RESIDENT_S.	
n.	
cs.	
pr.	

NON_RESI_S.	
n.	
cs.	
dr.	

STUDENT	
n.	
cs.	

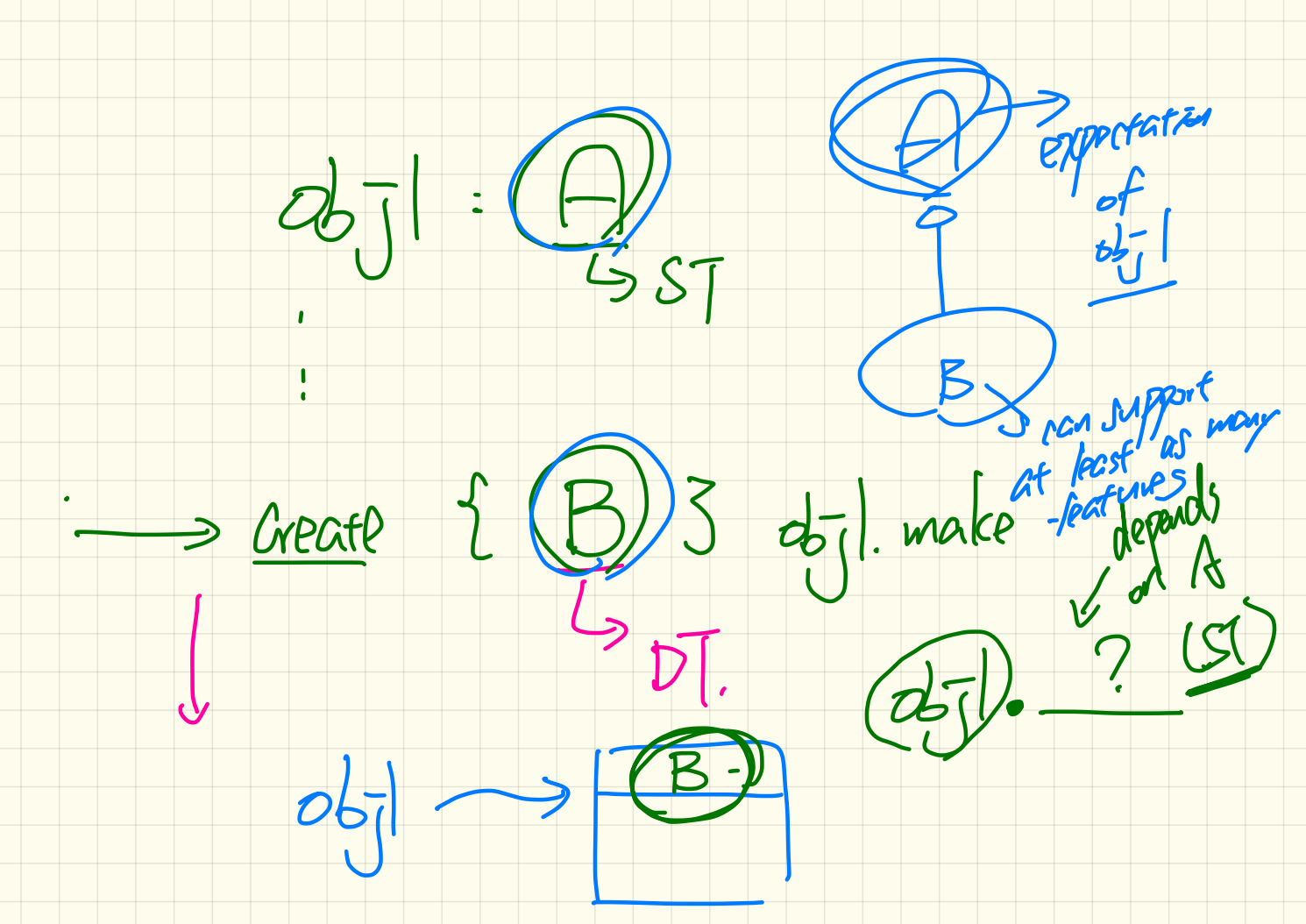
```
class STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
feature -- Commands that can be used as constructors.
  make (n: STRING) do name := n ; create courses.make end
feature -- Commands
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := 0.0
       across courses as c loop base := base + c.item.fee end
       Result := base
    end
end
```

```
test_dynamic_binding_students: BOOLEAN
  local
    jim: STUDENT
    rs: RESIDENT_STUDENT
    nrs: NON_RESIDENT_STUDENT
    c: COURSE
  do
    create c.make ("EECS3311", 500.0)
    create {STUDENT} jim.make ("J. Davis")
    create {RESIDENT_STUDENT} rs.make ("J. Davis")
    rs.register (c)
    rs.set_pr (1.5)
    jim := rs
    Result := jim.tuition = 750.0
    check Result end
    create {NON_RESIDENT_STUDENT} nrs.make ("J. Davis")
    nrs.register (c)
    nrs.set_dr (0.5)
    jim := nrs
    Result := jim.tuition = 250.0
end
```

COURSE	
t.	
fee	

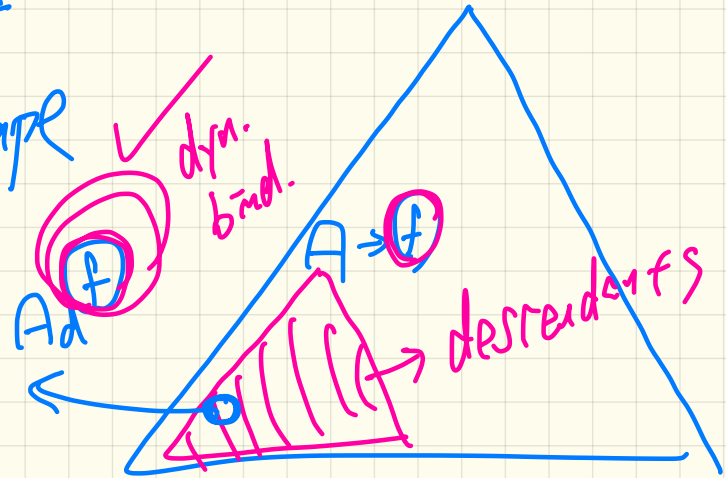
```
class
  RESIDENT_STUDENT
inherit
  STUDENT
    redefine tuition end
create make
feature -- Attributes
  premium_rate: REAL
feature -- Commands
  set_pr (r: REAL) do premium_rate := r end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := Precursor ; Result := base * premium_rate end
end
```

```
class
  NON_RESIDENT_STUDENT
inherit
  STUDENT
    redefine tuition end
create make
feature -- Attributes
  discount_rate: REAL
feature -- Commands
  set_dr (r: REAL) do discount_rate := r end
feature -- Queries
  tuition: REAL
    local base: REAL
    do base := Precursor ; Result := base * discount_rate end
end
```


obj1 : A
ST
A
expectation of obj1
B
can support at least as many features
depends on
obj1. ?
(ST)
Create { B }
obj1. make
DT.
obj1
B

Polymorphism

- multi
- shape

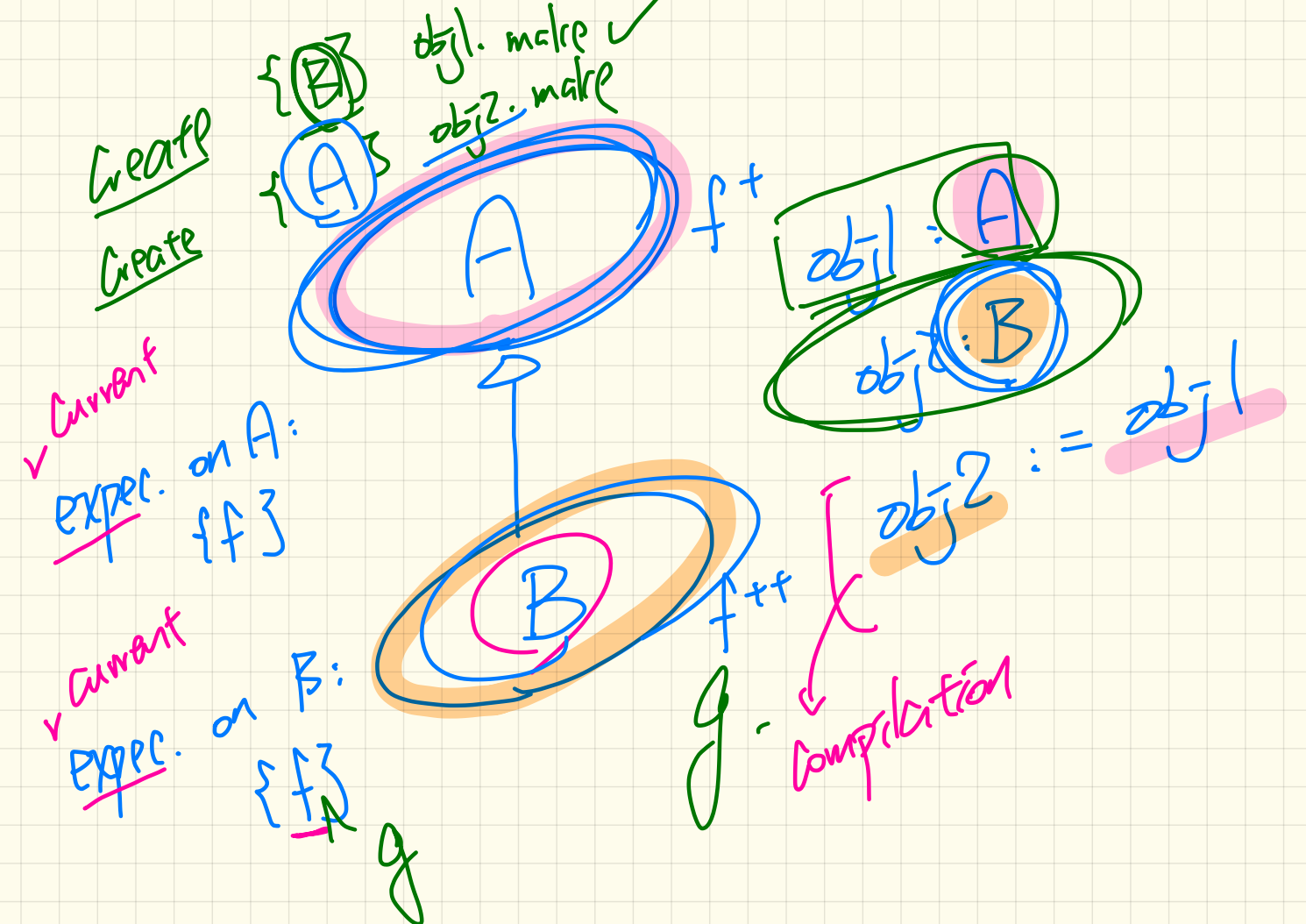


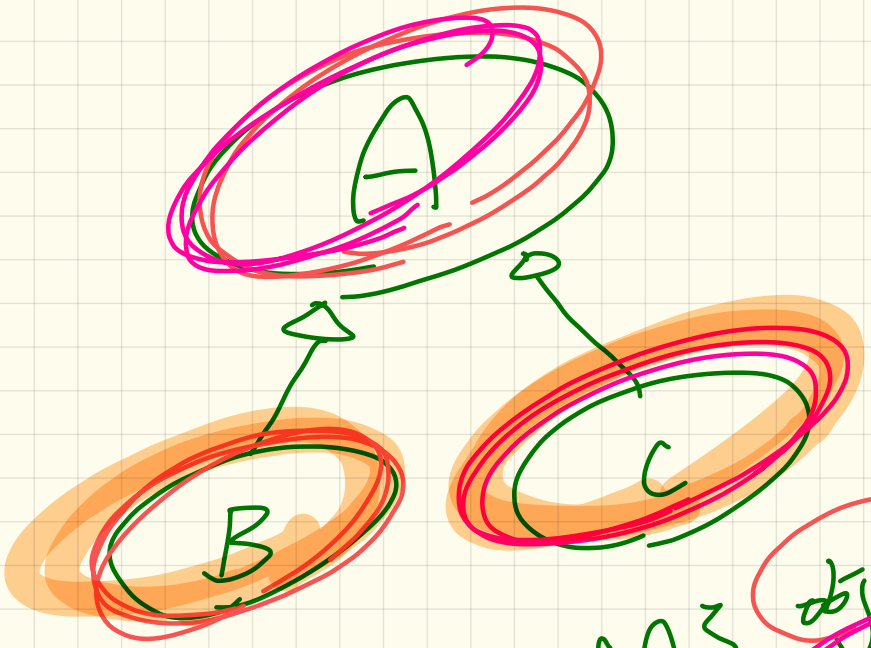
obj : (A) - (?) → any descendant class of A.

Create { Ad } (obj.) make(. ~)

obj.f

must support at least as many as A


obj1. make
obj2. make
Create
Create
{B}
{A}
A
f+
B
f++
g
obj1 : A
obj : B
obj2 := obj1
Current
expec. on A:
{f}
Current
expec. on B:
{f}
compilation



obj1 : A

obj2 : B

obj3 : C

Create {A}

Create {A}

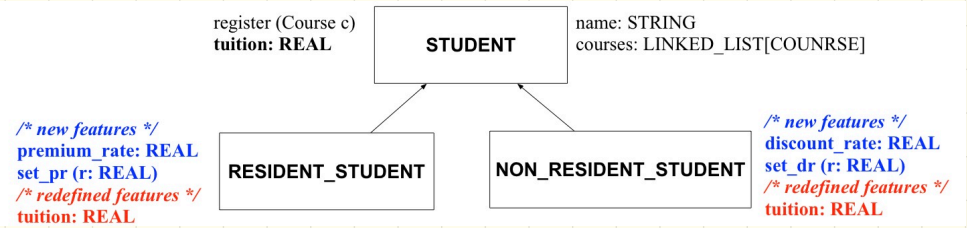
Create {B}

obj2.make(---)

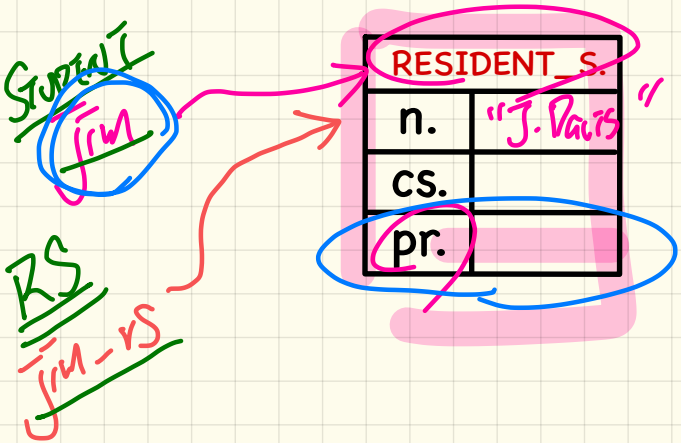
obj3.make(---)

obj3.make(---)

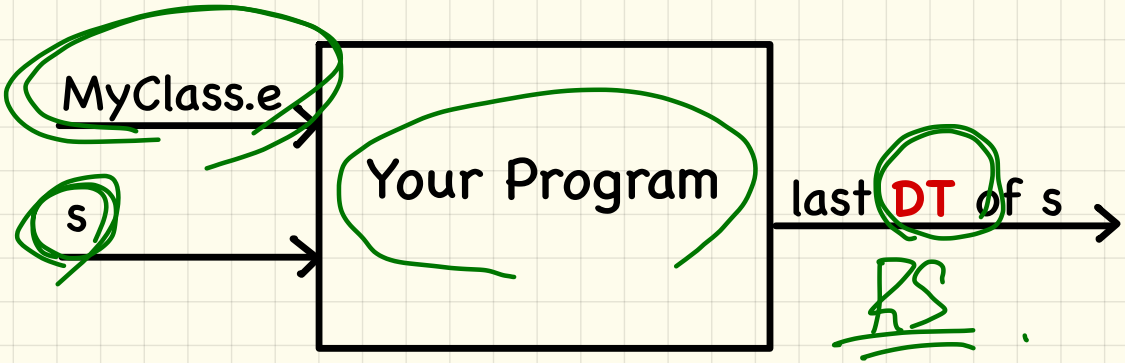
Type Cast: Motivation




```
1  local jim: STUDENT; rs: RESIDENT_STUDENT
2  do create {RESIDENT_STUDENT} jim.make ("J. Davis")
3    rs := jim
4    rs.setPremiumRate(1.5)
```



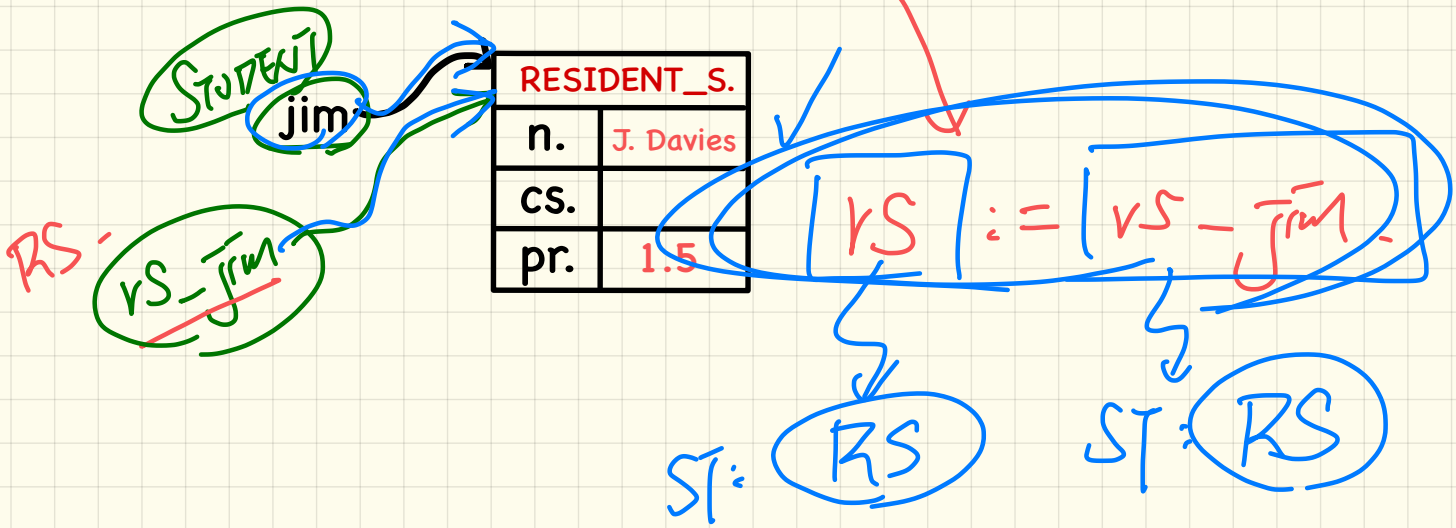
Inferring the DT of a Variable is Undecidable

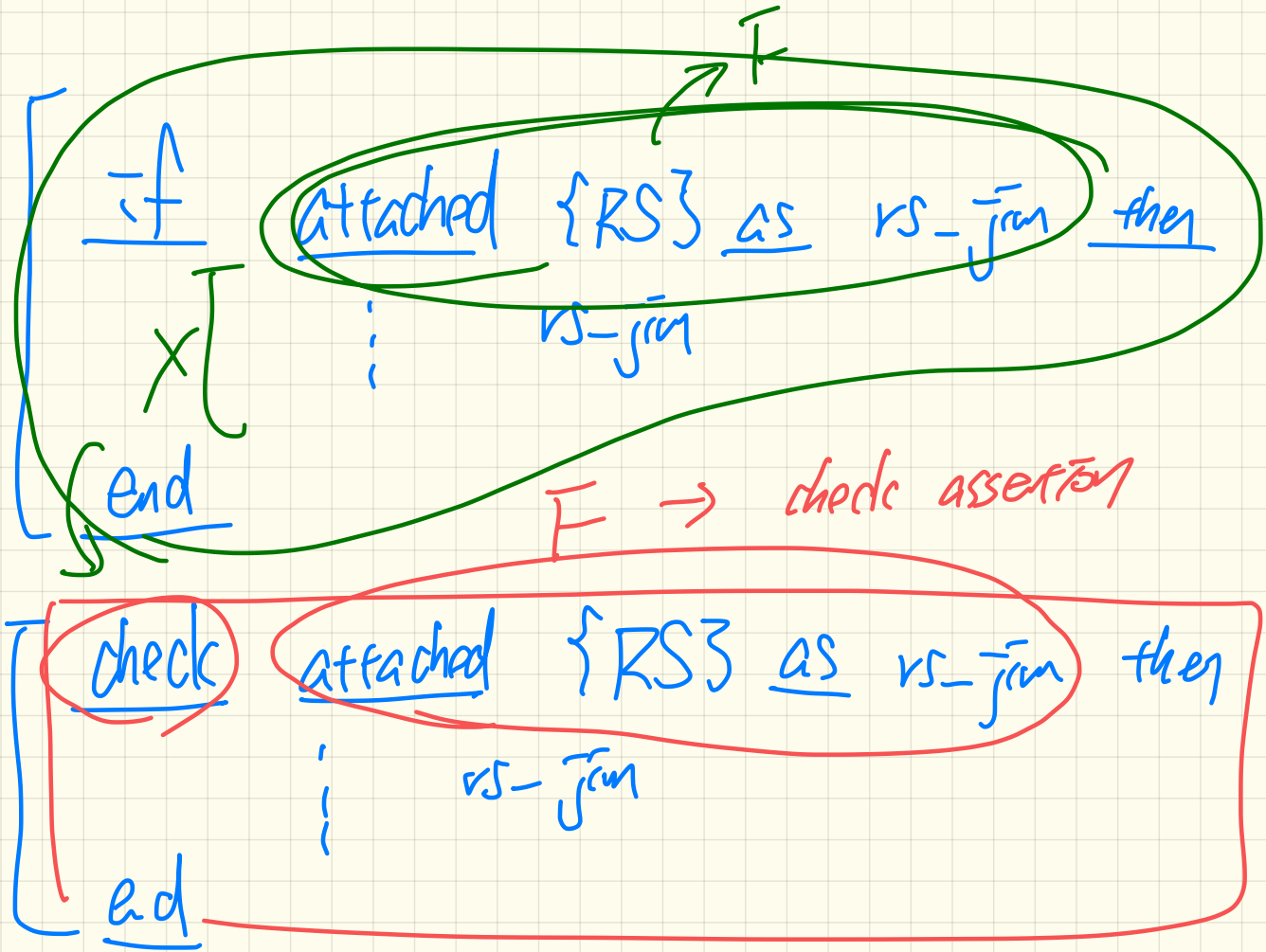



```
class MyClass
   make
      local
         s: STUDENT
      do
         create {RESIDENT_STUDENT} s.make
      end
end
```

Type Cast: Syntax

```
1  check attached {RESIDENT_STUDENT} jim as rs_jim then
2    rs := rs_jim
3    rs.set_pr (1.5)
4  end
```




T
if
attached {RS} as rs_jim then
rs_jim
x
end
F → check assertion
check
attached {RS} as rs_jim then
rs_jim
end