LECTURE 10

TUESDAY OCTOBER 8

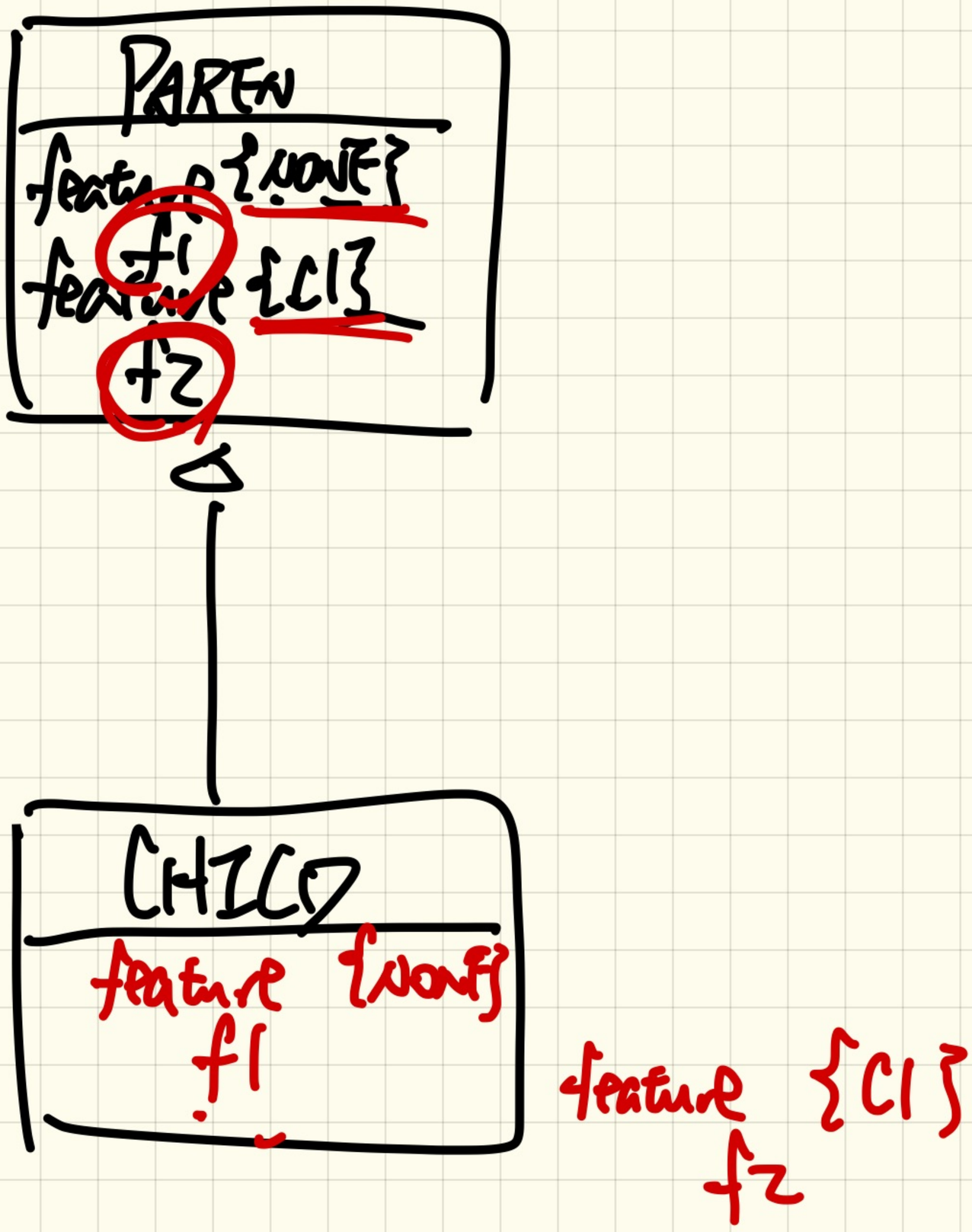
Violation of the Single Choice Principle

```
class RESIDENT_STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
  premium_rate: REAL
feature -- Constructor
  make (n: STRING)
    do name := n ; create courses.make end
feature -- Commands
  set_pr (r: REAL) do premium_rate := r end
  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 * premium_rate
    end
end
```

```
class NON_RESIDENT_STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
  discount_rate: REAL
feature -- Constructor
  make (n: STRING)
    do name := n ; create courses.make end
feature -- Commands
  set_dr (r: REAL) do discount_rate := r end
  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 * discount_rate
    end
end
```

cmd (...)
do
Precursor (...)
end

f (...): INT
do
Result :=
Precursor (...)
end


PARENT
feature {MONEY}
f1
feature {CI}
f2
CHILD
feature {wonf}
f1
feature {CI}
f2

Static Type vs. Dynamic Type

- In Java:

```
Student s = new Student("Alan");
Student rs = new ResidentStudent("Mark");
```

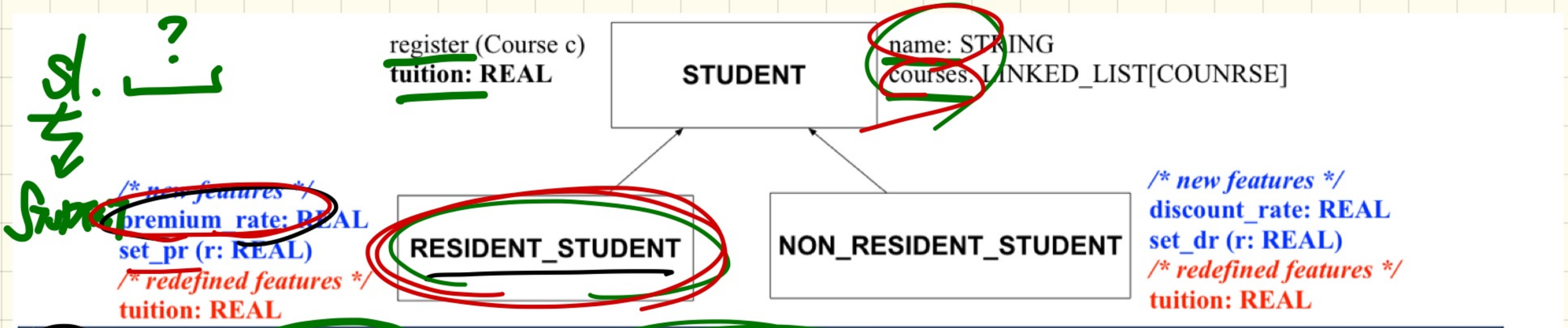
- In Eiffel:

```
local s: STUDENT
      rs: STUDENT
do create {STUDENT} s.make ("Alan")
   create {RESIDENT_STUDENT} rs.make ("Mark")
```

  - In Eiffel, the *dynamic type* can be omitted if it is meant to be the same as the *static type*:

```
local s: STUDENT
do create s.make ("Alan")
```

Student Classes (with Inheritance): Expectations




```
s1, s2, s3: STUDENT ; rs: RESIDENT_STUDENT ; nrs : NON_RESIDENT_STUDENT
create {STUDENT} s1.make ("S1")
create {RESIDENT_STUDENT} s2.make ("S2")
create {NON_RESIDENT_STUDENT} s3.make ("S3")
create {RESIDENT_STUDENT} rs.make ("RS")
create {NON_RESIDENT_STUDENT} nrs.make ("NRS")
```

create {RS} s.make

	name	courses	reg	tuition	pr	set_pr	dr	set_dr
s1	✓	✓	✓	✓	x	x	x	x
s2	✓	✓	✓	✓	x	x	x	x
s3								
rs	✓	✓	✓	✓	✓	✓	x	x
nrs								

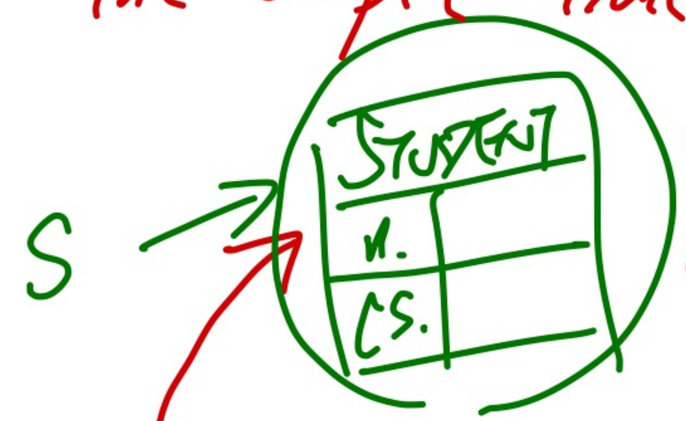
Polymorphism: Intuition

```
1  local
2    s: STUDENT
3    rs: RESIDENT_STUDENT
4  do
5    create s.make ("Stella")
6    create rs.make ("Rachael")
7    rs.set_pr (1.25)
8    s := rs  /* Is this valid? */
9    rs := s  /* Is this valid? */
```

cs

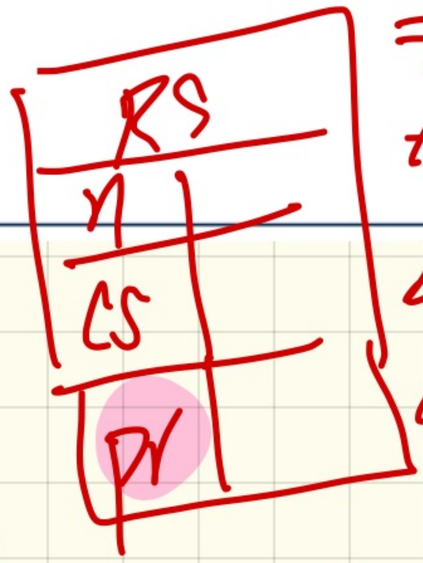
ST of rs

if this was allowed at the compile time, we would get a runtime crash



s

rs



⇒ this should be a compilation error.

Assume rs := s was allowed

What can expect to call on rs?

rs.pr

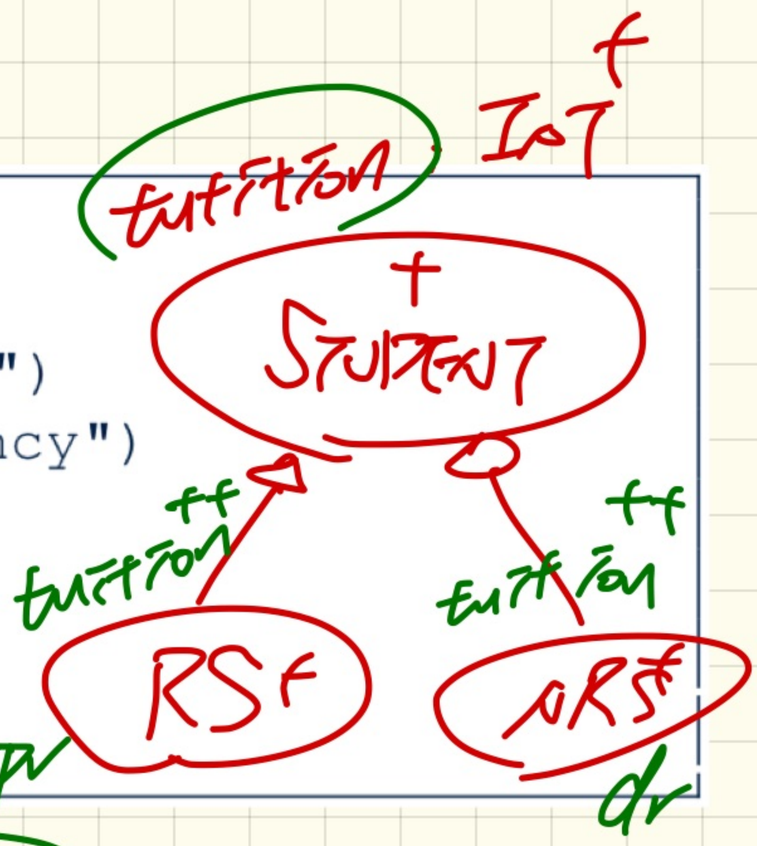
rs.set_pr

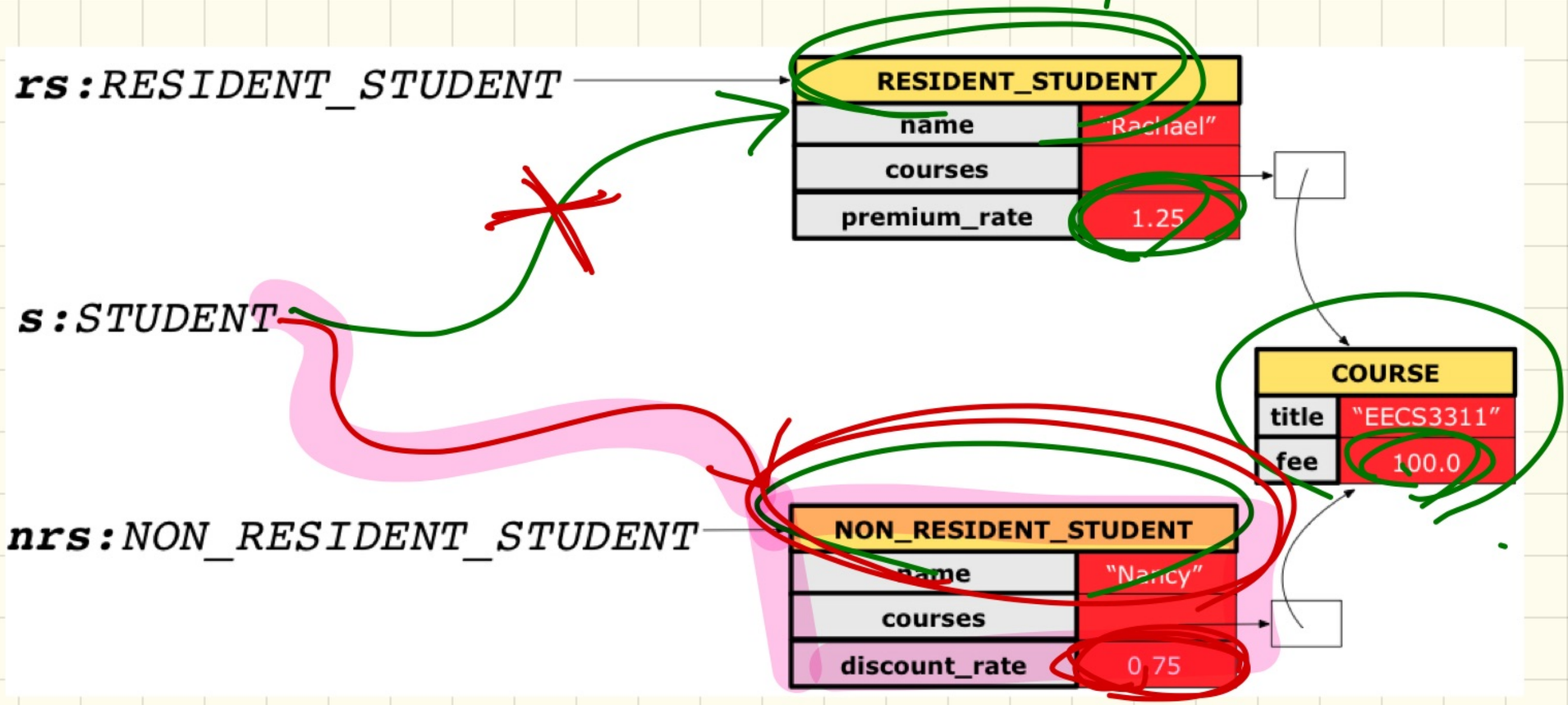
Crash ‘i no pr on a STUDENT object

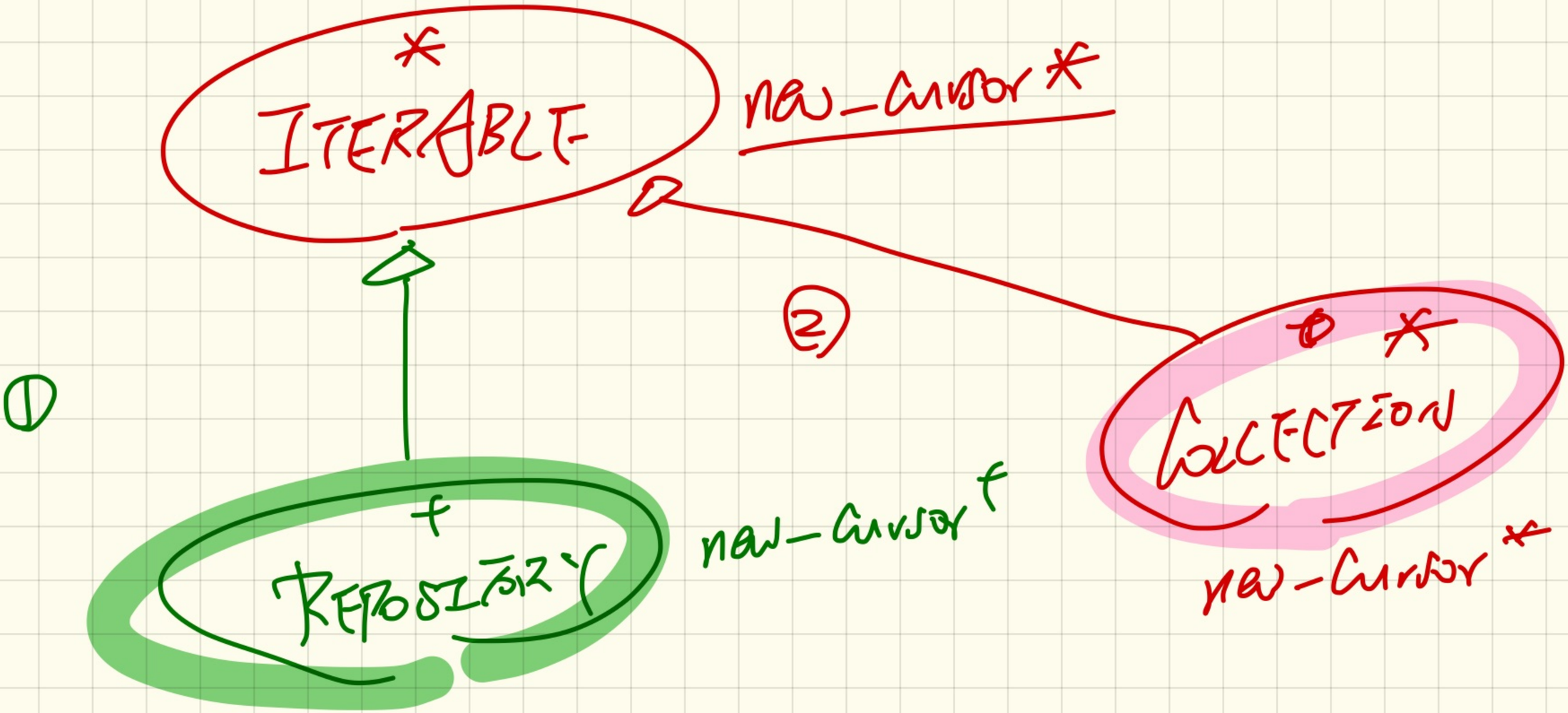
c') rs := s not allowed.

Dynamic Binding: Intuition

```
1  local c : COURSE ; s : STUDENT
2  do crate c.make ("EECS3311", 100.0)
3   create {RESIDENT_STUDENT} rs.make("Rachael")
4   create {NON_RESIDENT_STUDENT} nrs.make("Nancy")
5     rs.set_pr(1.25); rs.register(c)
6     nrs.set_dr(0.75); nrs.register(c)
7     s := rs ; check s.tuition = 125.0 end
8     s := nrs ; check s.tuition = 75.0 end
```






*
ITERABLE
new_cursor *
1
2
f
REPOSITORY
new_cursor f
0 *
COLLECTION
new_cursor *

model

ACCOUNT

feature -- *Commands*
withdraw (amount: **INTEGER**)
require
non_negative_amount: amount > 0
affordable_amount: amount ≤ balance
do
balance := balance - amount
ensure
balance_deduced: balance = **old** balance - amount
end

tests

TEST_ACCOUNT

feature -- *Test Commands for Contract Violations*
test_withdraw_postcondition_violation
local
acc: BAD_ACCOUNT_WITHDRAW
do
create acc.make ("Alan", 100)
-- *Violation of Postcondition*
-- *with tag "balance_deduced" expected*
acc.*withdraw* (50)
end

acc

BAD_ACCOUNT_WITHDRAW

feature -- *Redefined Commands*
withdraw (amount: **INTEGER**) ++
do
Precursor (amount)
-- *Wrong Implementation*
balance := balance + 2 * amount
end

Adding Postcondition Tests

LAB 2

1. encourage to test postconditions
2. do not submit the new classes that you created

```
1  class TEST_ACCOUNT
2  inherit ES_TEST
3  create make
4  feature -- Constructor for adding tests
5    make
6      do
7        add_violation_case_with_tag ("balance_deducted",
8          agent test_withdraw_postcondition_violation)
9      end
10 feature -- Test commands (test to fail)
11   test_withdraw_postcondition_violation
12     local
13       acc: BAD_ACCOUNT_WITHDRAW
14     do
15       comment ("test: expected postcondition violation of withdraw")
16       create acc.make ("Alan", 100)
17       -- Postcondition Violation with tag "balance_deduced" to occur.
18       acc.withdraw (50)
19     end
20 end
```

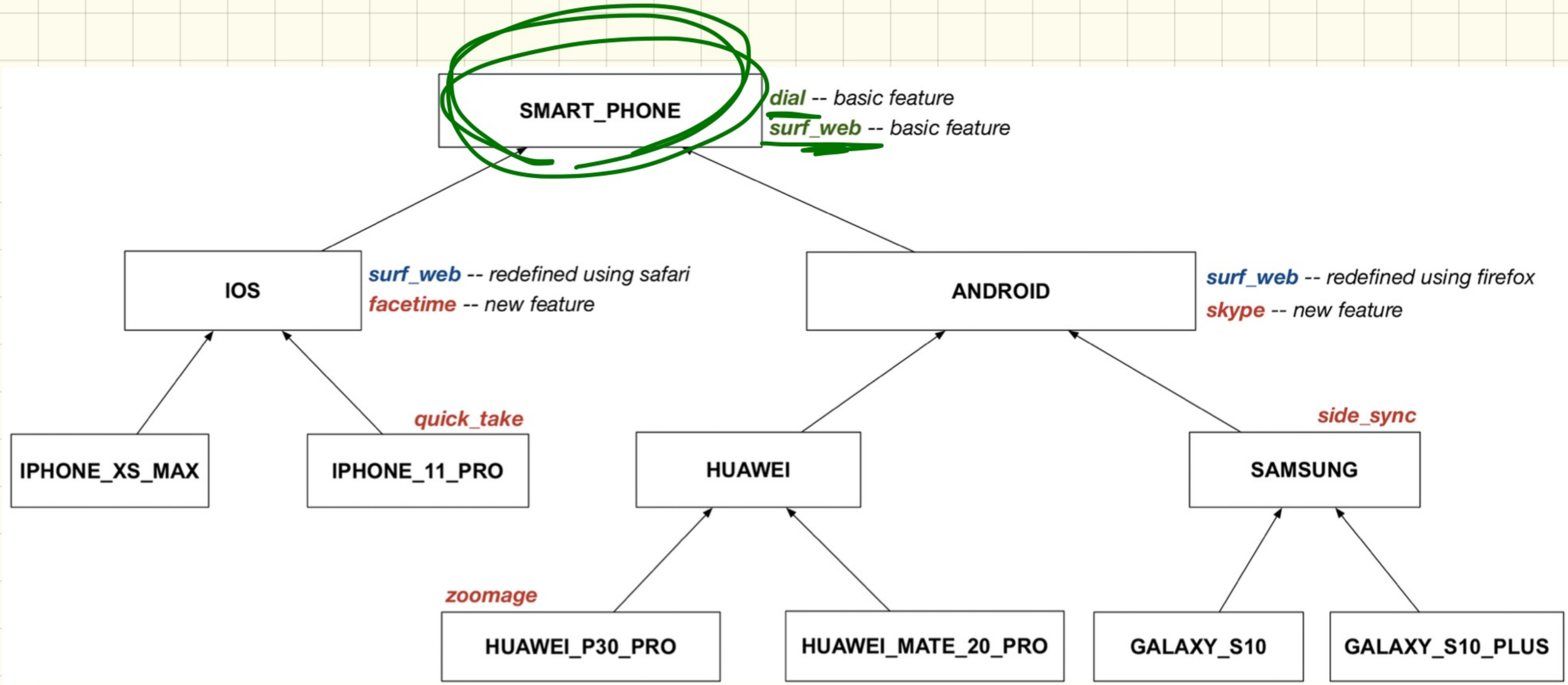
Testing of Postcondition: Exercise

```
class BANK
  deposit_on_v5 (n: STRING; a: INTEGER)
    do ... -- Put Correct Implementation Here.
    ensure
      ...
      others_unchanged :
        across old accounts.deep_twin as cursor
        all cursor.item.owner /~ n implies
            cursor.item ~ account_of (cursor.item.owner)
        end
    end
end
```

TEST

```
class BAD_BANK_DEPOSIT
inherit BANK redefine deposit end
feature -- redefined feature
  deposit_on_v5 (n: STRING; a: INTEGER)
    do Precursor (n, a)
       accounts[accounts.lower].deposit(a)
    end
end
```

Multi-Level Inheritance Hierarchy of Smartphones



jim : Student

rs : RS

rs := jim

No. ? ST of jim (Student) is not a descendant class of the ST of rs (RS)

S : STUDENT

S. tuition

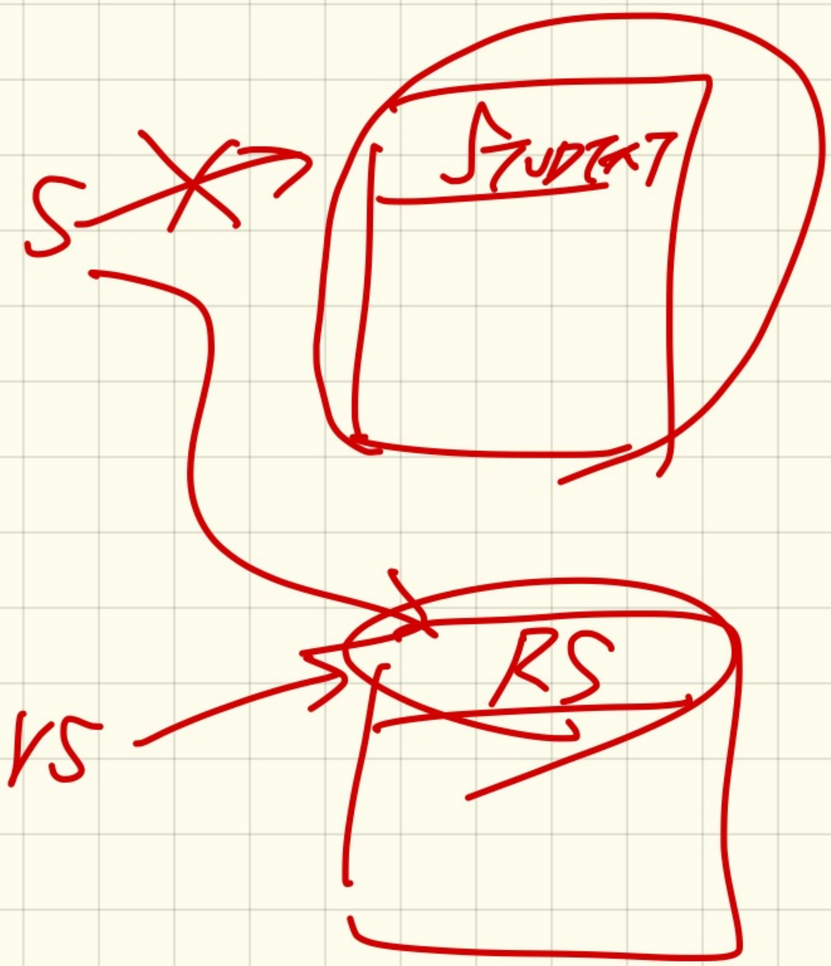
↳ STUDENT

RS

NRS

S : STUDENT

rs : RS

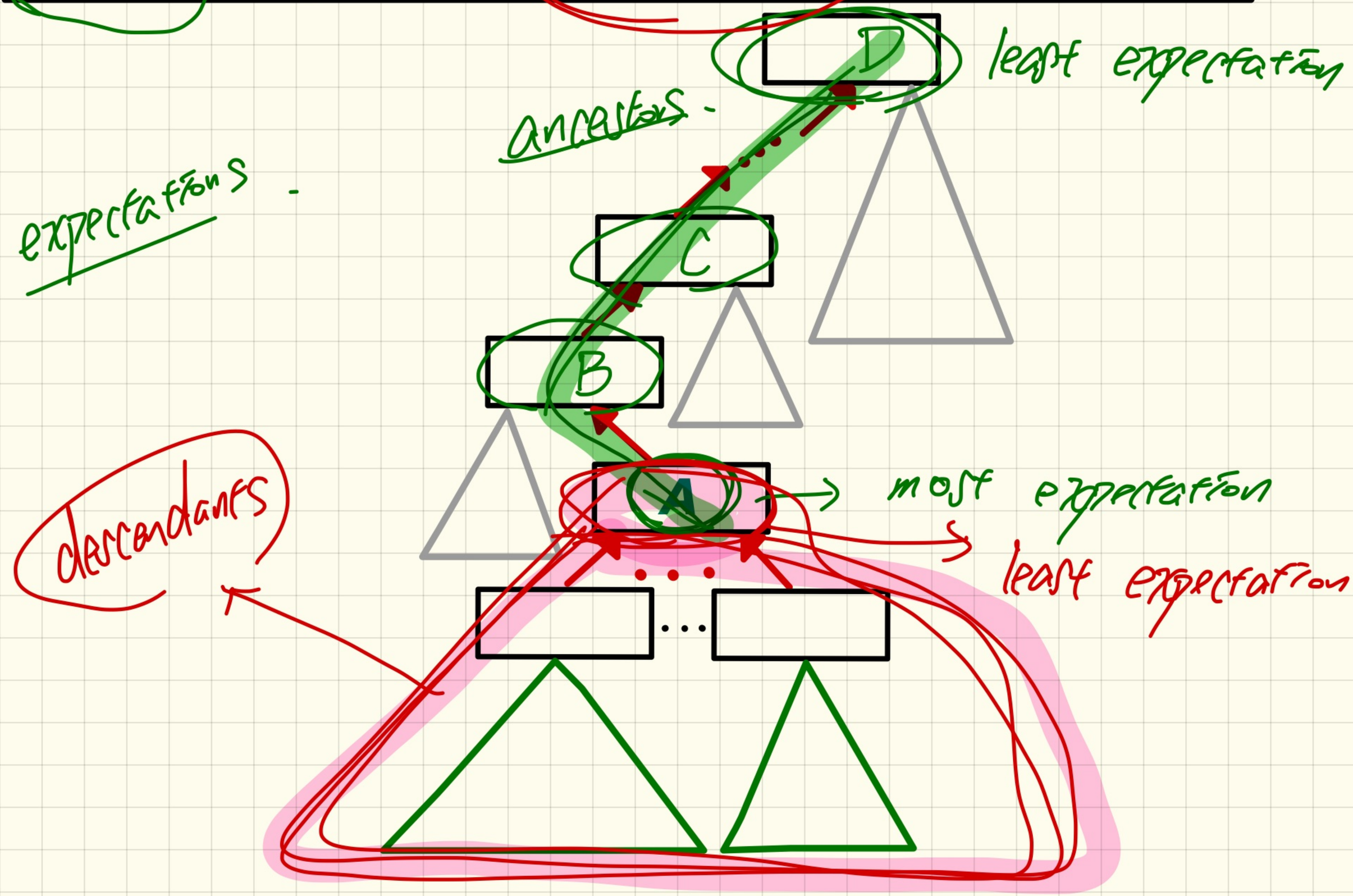


Create s. make

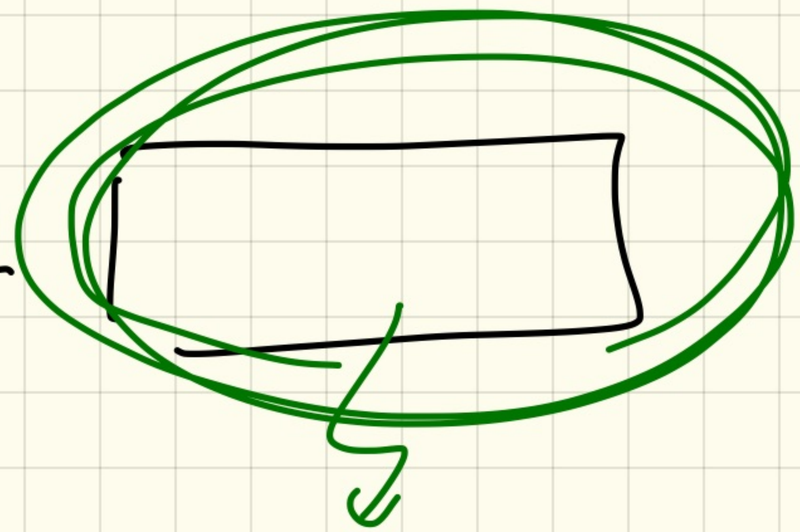
Create vs. make

S := rs

Ancestors, Expectations, Descendants, and Code Reuse



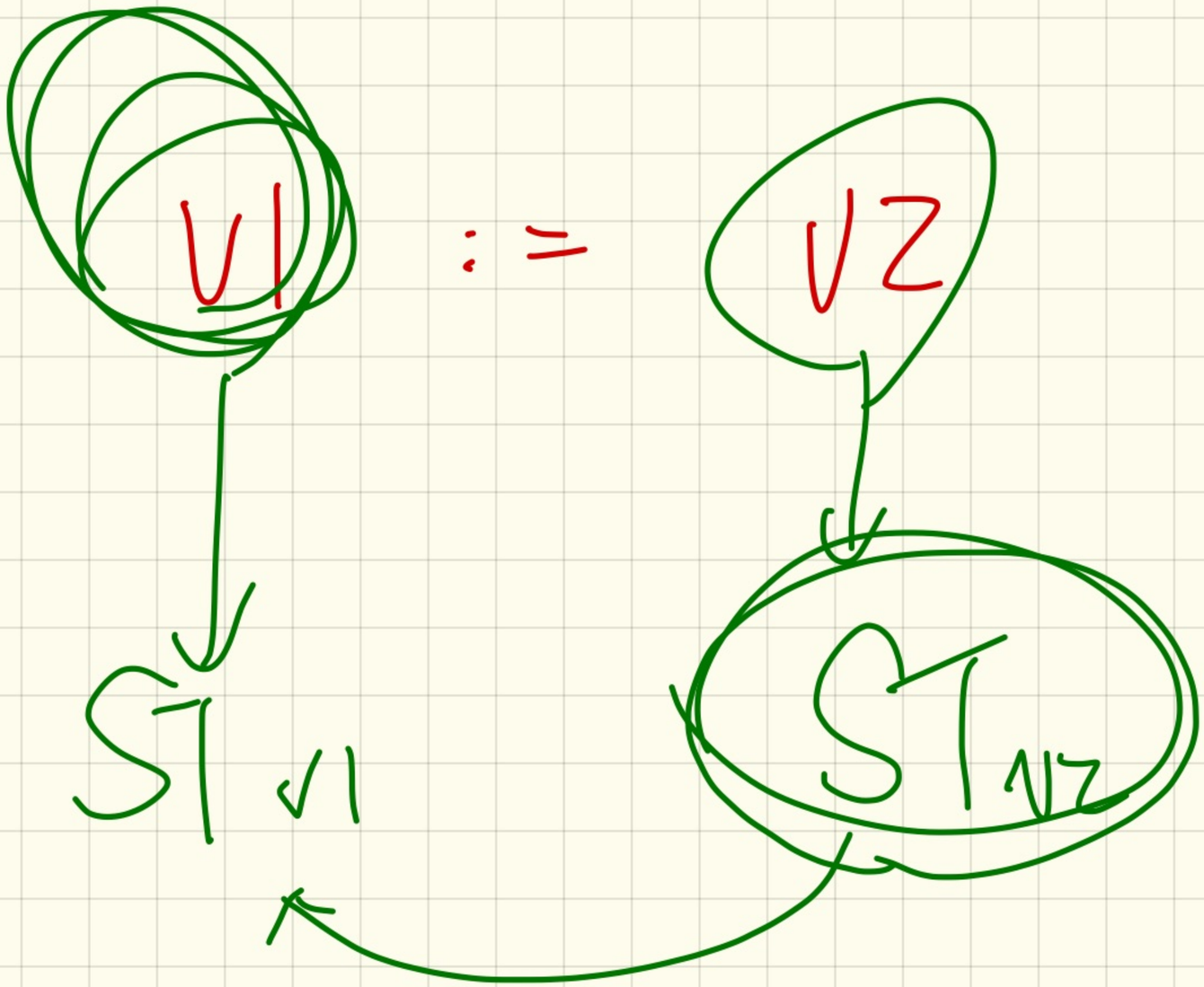
S : STUDENT.

S' := 

S.n

S.CS

any expression whose static type is a descendant of the ST of S.



a descendant class of ST_{V1}?

S: STUDENT

S. tuition

look up the ST of S:

↳ does function exist in that ST?