Lecture 15
Monday March 2

- Office Hours Change (for the rest of the term)
  Monday's office hours moved to:
  12:30 to 14:30 on Tuesdays

- Lab 4 due 3pm Friday March 13

- Automated regression testing

Design Attempt 2.5

```
class
   STUDENT
create
   make
feature -- kind-specific attributes
   rate: LINKED_LIST[REAL]
   tuition: LINKED_LIST[REAL]
   max: LINKED_LIST[INTEGER]
feature -- attributes
   courses: LINKED_LIST[COURSE]
   kind: INTEGER
feature -- command
   make (kind: INTEGER)
      do
         kind := a_kind
         rate := << 1.25, 0.75 >>
         max := << 6, 4 >>
         tuition := <<0.0, 0.0 >>
      end
   ...
end
```

```
get_tuition: REAL
   do
      across courses is c loop
         tuition[kind] :=
            tuition[kind] + c.fee
      end
      tuition[kind] :=
         tuition[kind] * rate[kind]
   end
```

```
register (c: COURSE)
   do
      if courses.count = MAX then -- Error
      else courses.extend (c)
      end
   end
```

Good design?

Judge by Single Choice Principle

- A new kind is introduced?
- An existing kind is obeselete?

Desigh 3:

Inheritance

Code Reuse

Cohesion?

Single Choice Principle?

Collection of Students?

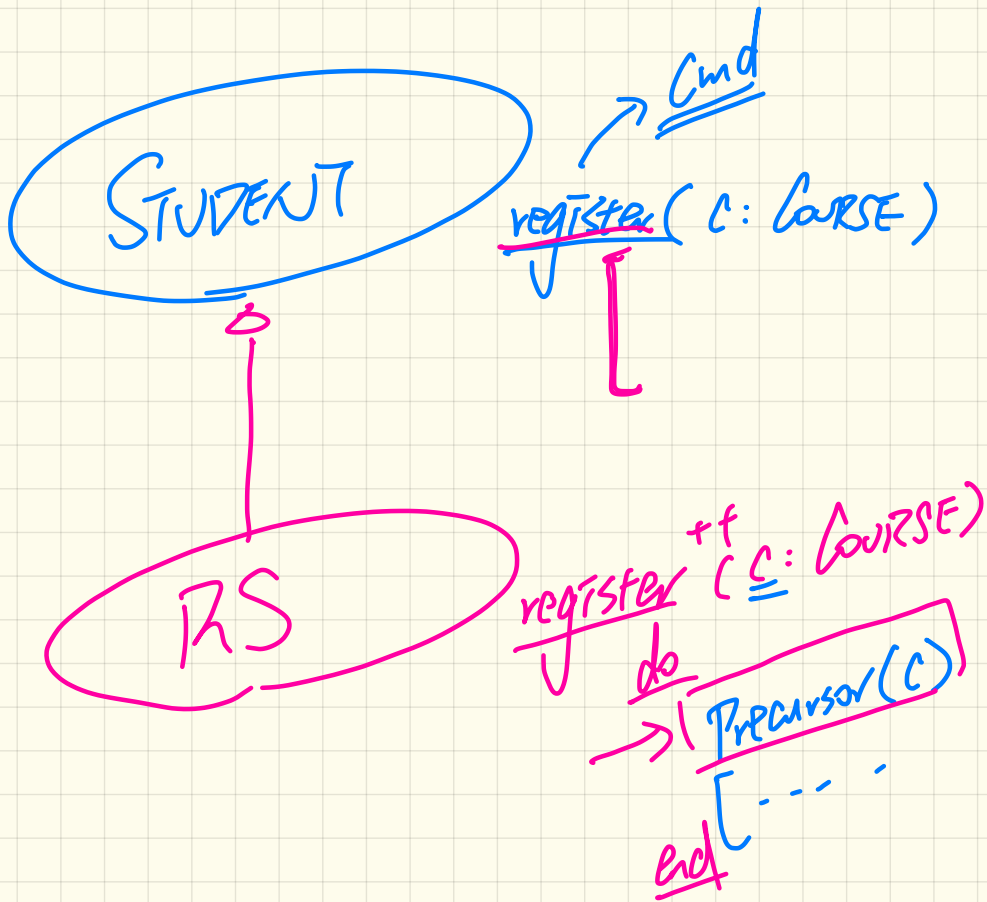
```
class STUDENT
create make
feature -- Attributes
  name: STRING
  courses: LINKED_LIST[COURSE]
feature -- Commands that can be used as constructors.
  make (n: STRING) do name := n ; create courses.make end
feature -- Commands
  register (c: COURSE) do courses.extend (c) end
feature -- Queries
  tuition: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

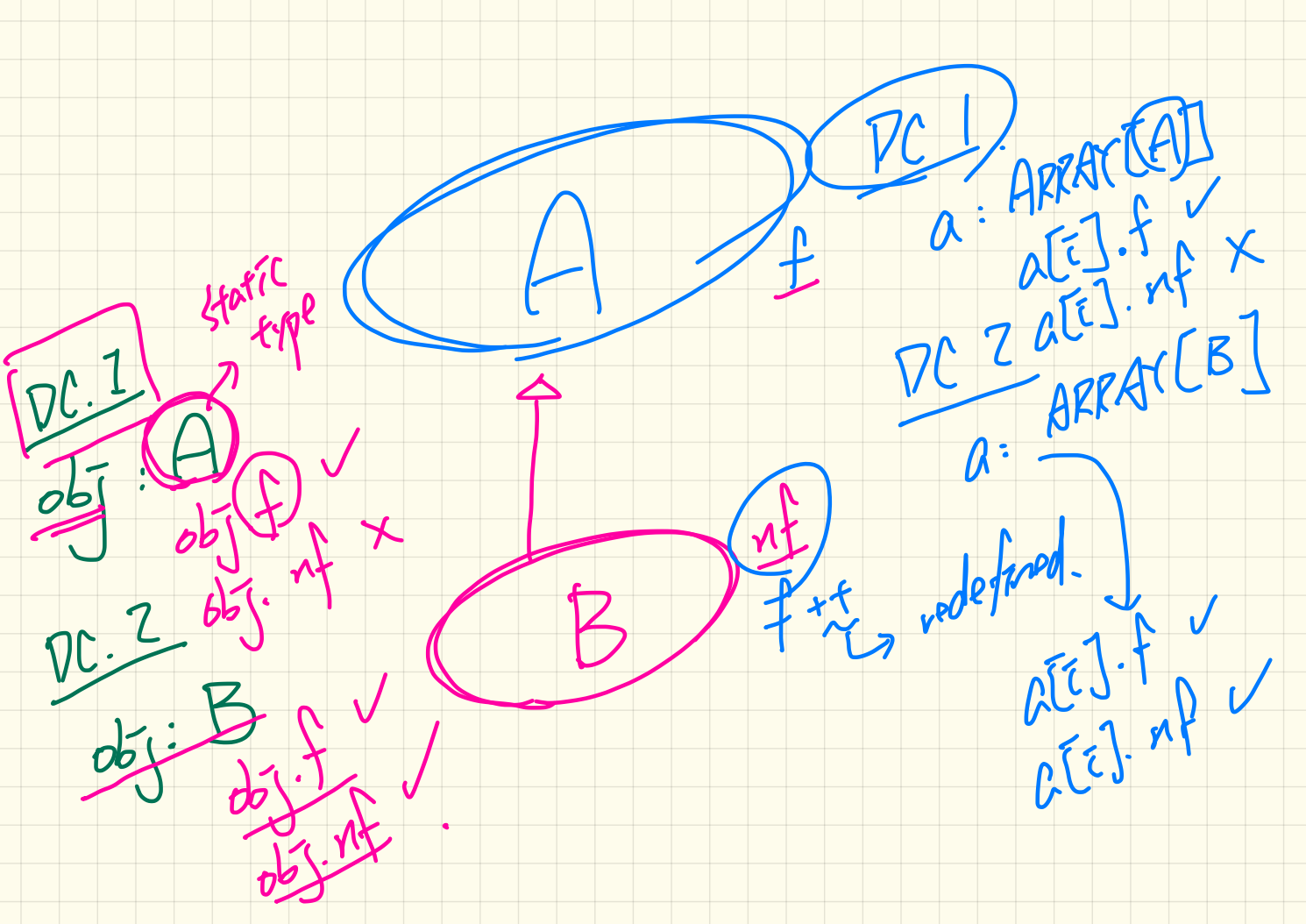
Precursor

```
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
```

version of tuition for the parent class

```
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
```


STUDENT
RS
Cmd
register (c : COURSE)
register +f (c : COURSE)
do
precursor (c)
end


A
B
f
nf
DC 1
a : ARRAY[A]
a[i].f ✓
a[i].nf ✗
DC 2 a[i].nf ✗
a : ARRAY[B]
redefined.
a[i].f ✓
a[i].nf ✓
DC.1
obj : A
static type
obj.f ✓
obj.nf ✗
DC.2
obj : B
obj.f ✓
obj.nf ✓

Desigh 3:

Inheritance

Code Reuse

Cohesion?

Single Choice Principle?

Collection of Students?

```
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
```

```
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
```

Desigh 3:

Inheritance

Code Reuse

Cohesion?

Single Choice Principle?

Collection of Students?

```
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
```

```
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
```

With Inheritance (Design 3) Collection of Students

```
class
   STUDENT_MANAGEMENT_SYSTEM
feature -- attributes
   students: LINKED_LIST[STUDENT]
feature -- command
   add_student(s: STUDENT)
      do
         students.extend(s)
      end
   register_all (c: COURSE)
     do
       across students is s
          loop
             s.register(c)
          end
     end
end
```

```
c: COURSE
rs: STUDENT
nrs: STUDENT
sms: SMS
create c.make("3311")
create sms.make




sms.add_student(rs)
sms.add_student(nrs)
sms.register_all(c)
```

Q: What if more kinds of students are to be introduced?

Static Type vs. Dynamic Type

- In Java:

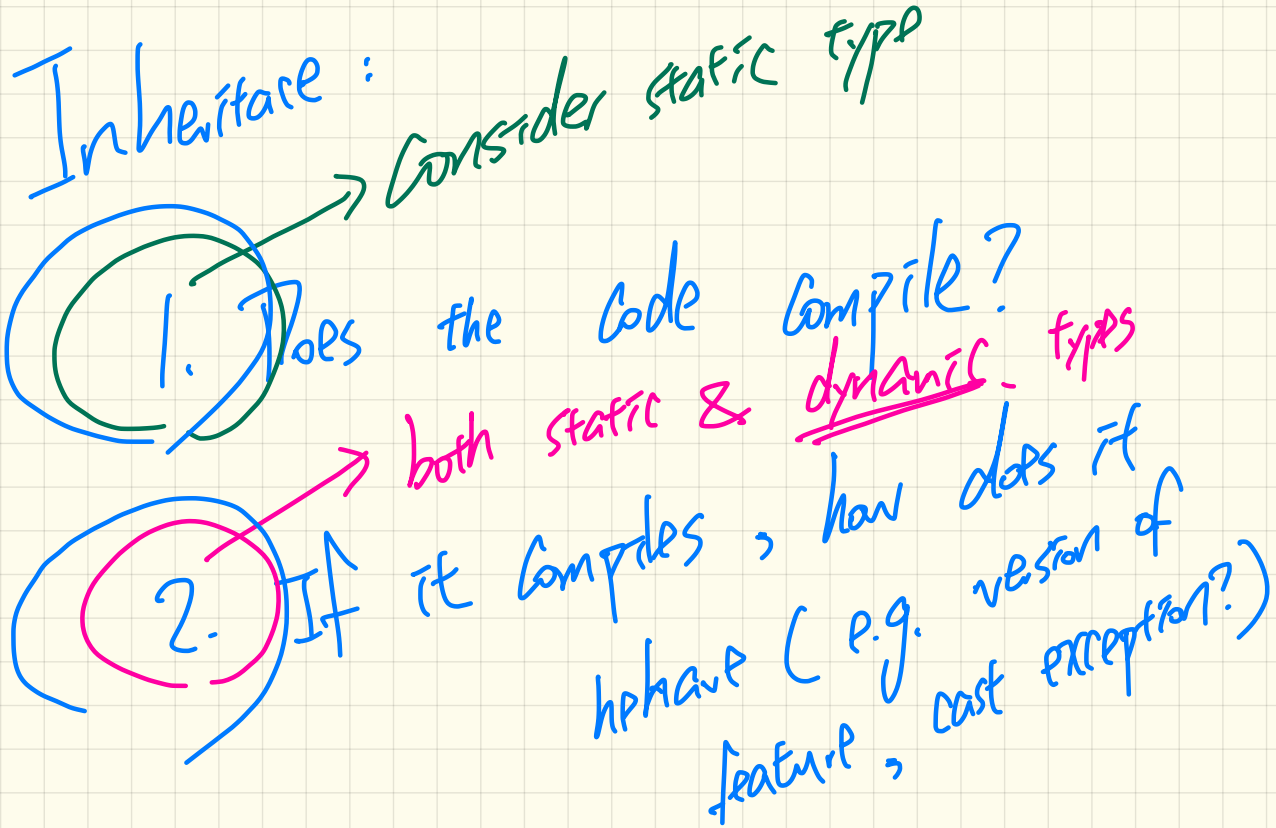
```
Student s = new Student("Alan");
Student rs = new ResidentStudent("Mark");
```

- In Eiffel:

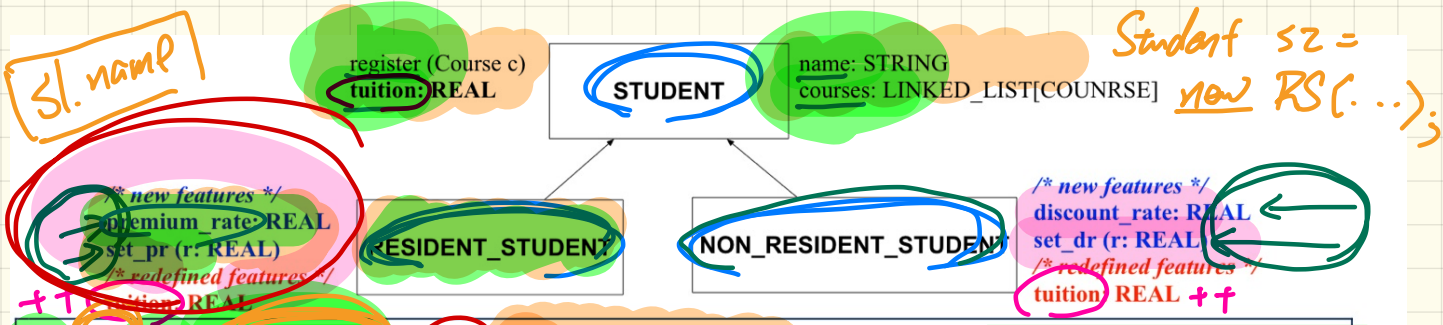
```
local s: STUDENT
      rs: STUDENT
do create {STUDENT} s.make ("Alan")
   create {RESIDENT_STUDENT} rs.make ("Mark")
```

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

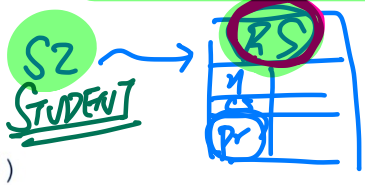
```
local s: STUDENT
do create s.make ("Alan")
```


Inheritance:
Consider static type
1. Does the code compile?
2. If it compiles, both static & dynamic types
how does it behave (e.g. version of feature, cast exception?)

Student Classes (with Inheritance): Expectations

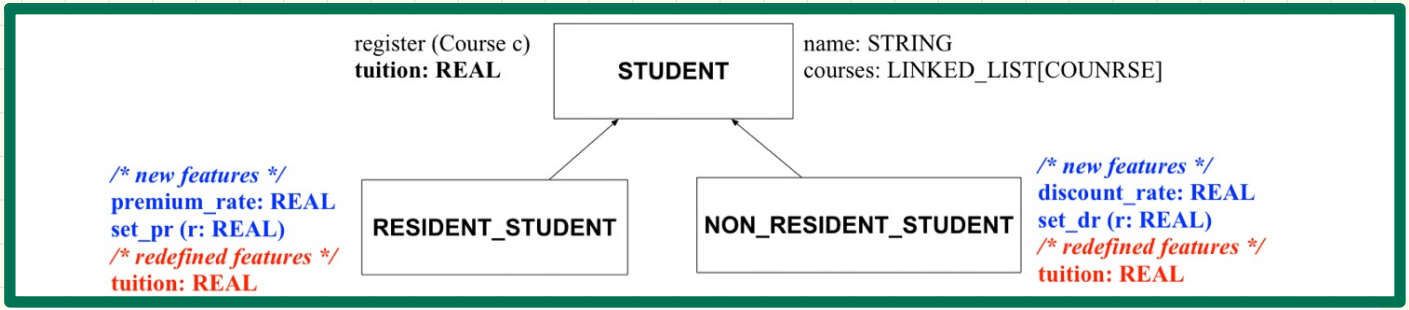



```
s1, s2, s3: STUDENT ; rs: RESIDENT_STUDENT ; nrs : NON_RESIDENT_STUDENT
create {STUDENT} s1.make ("S1")
create {RESIDENT_STUDENT} s2.make ("S2")
create {NON_RESIDENT_STUDENT} s3.make ("S3")
create {RESIDENT_STUDENT} rs.make ("RS")
create {NON_RESIDENT_STUDENT} nrs.make ("NRS")
```

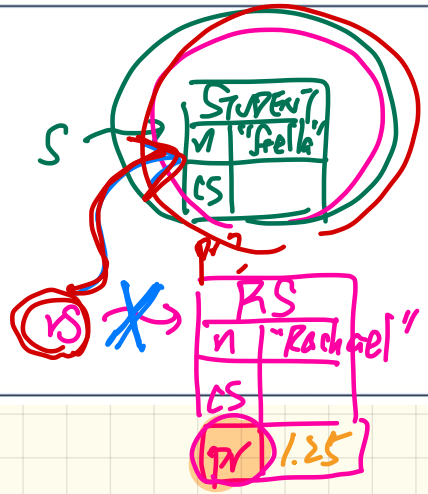


	name	courses	reg	tuition	pr	set_pr	dr	set_dr
s1								
s2								
s3								
rs								
nrs								

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? */
```

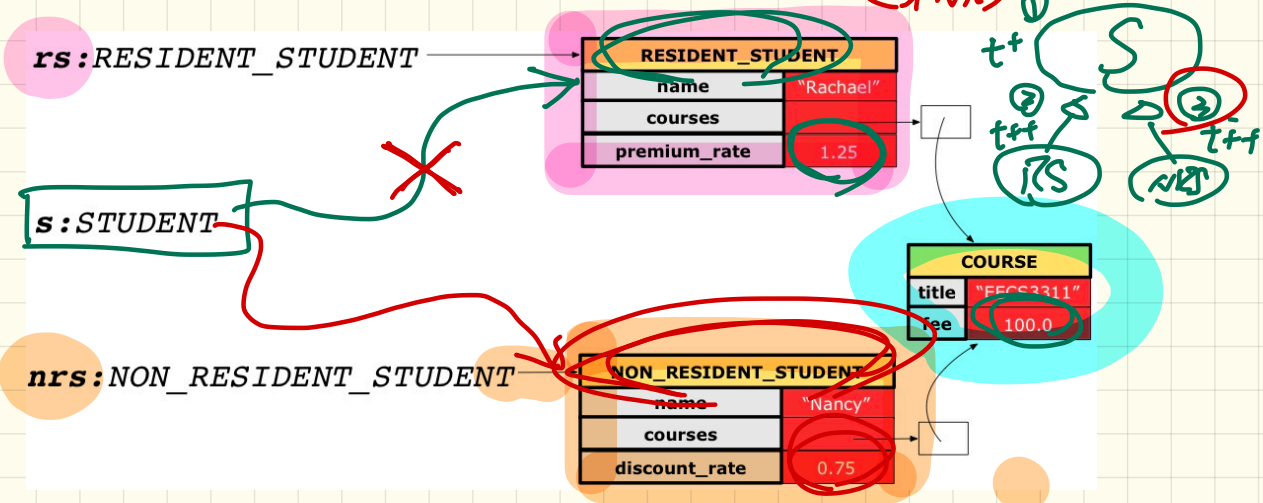


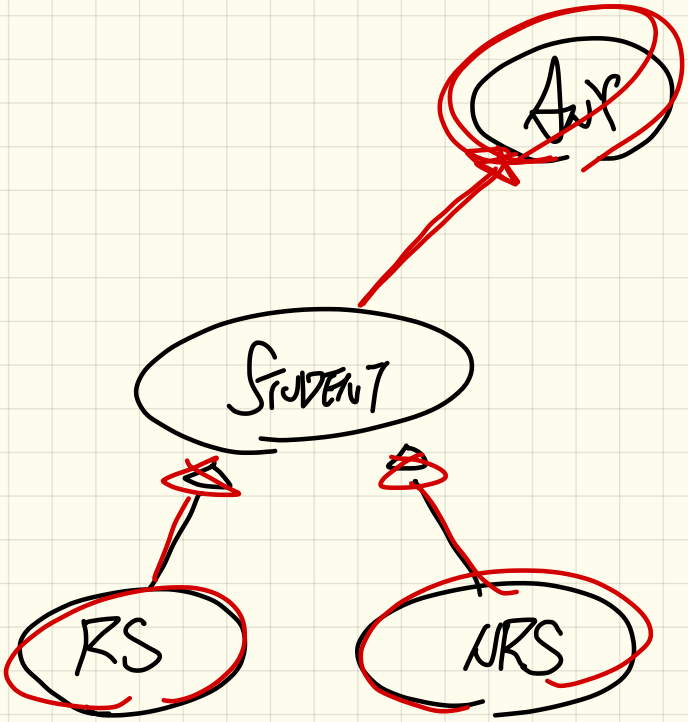
Proof by Contradiction

1. Assume 9 compiled.
2. ⇒ rs points to the STUDENT object
3. Expectations on rs: name, pr, set_pr
4. rs.pr ⇒ Crash.

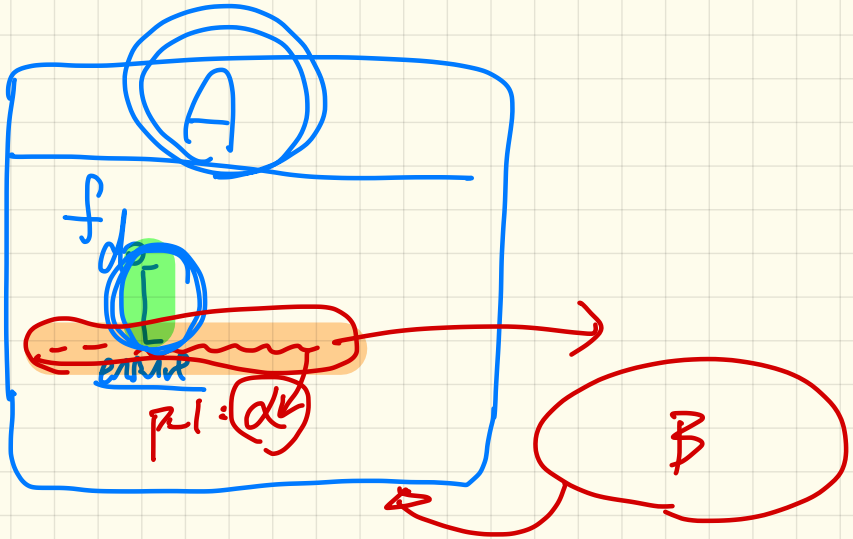
Dynamic Binding: Intuition

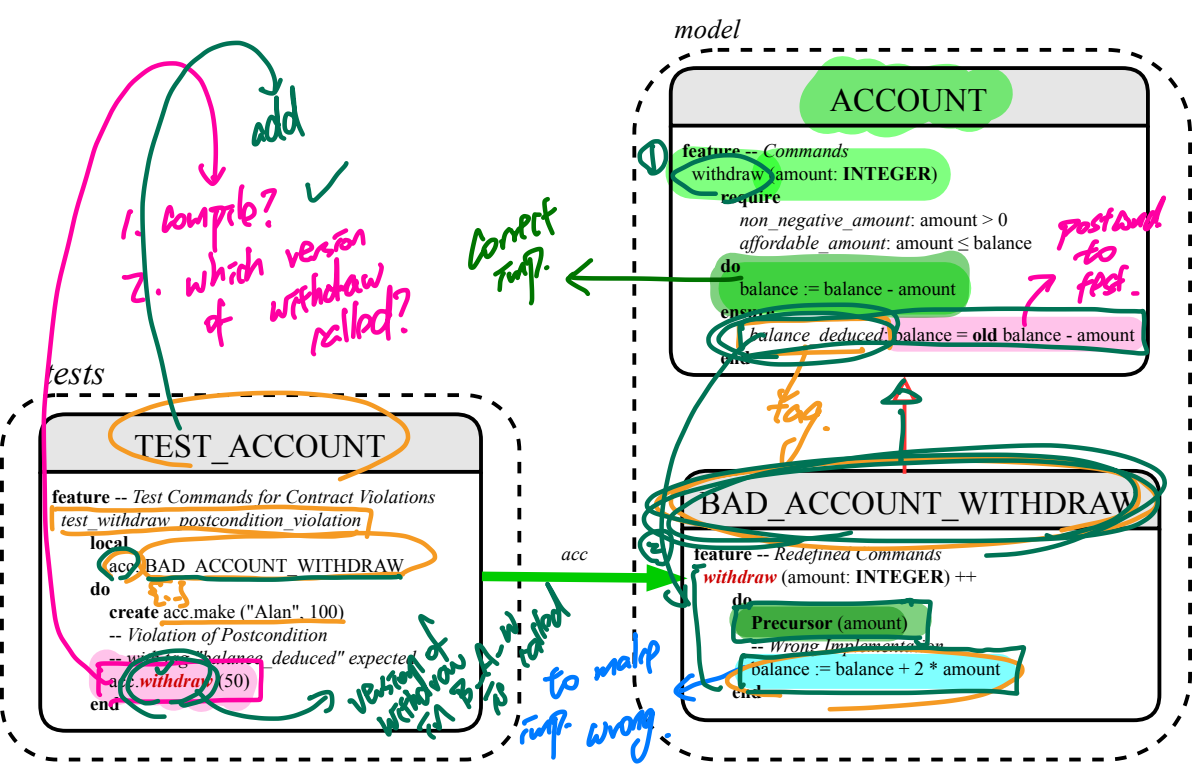
```
1  local c : COURSE ; s : STUDENT
2  do crate c.make ("EECS3311", 100.0)
3     create {RESIDENT_STUDENT} rs.make("Rachael")
4     create {NON_RESIDENT_STUDENT} nrs.make("Nancy")
5     rs.set_pr(1.25); rs.register(c)
6     nrs.set_dr(0.75); nrs.register(c)
7     s := rs ; check s.tuition = 125.0 end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




Aux
Student
RS
NRS

Testing of Postcondition.




model
ACCOUNT
feature -- Commands
withdraw (amount: INTEGER)
require
non_negative_amount: amount > 0
affordable_amount: amount ≤ balance
do
balance := balance - amount
ensure
balance_deducted: 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
add
1. Compile? ✓
2. which version of withdraw called?
Correct Impl.
post cond. to test.
tag.
version of withdraw for BAD_W is called
to make impl. wrong.

Adding Postcondition Tests

```
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
```


```
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
```