Monday February 25

Lecture 12

Static Type vs. Dynamic Type

- In Java:

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

- In Eiffel:

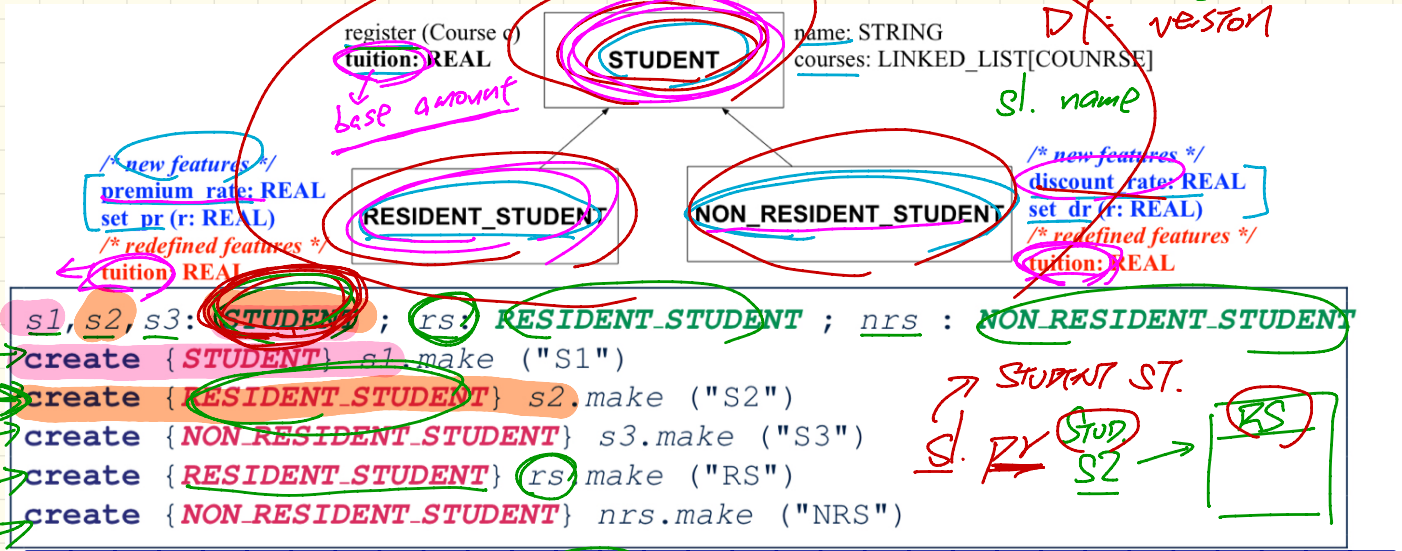
```
local s: STUDENT
      rs: STUDENT
do create {STUDENT} s.make ("Alan")
   create {RESIDENT_STUDENT} rs.make ("Mark")
```

  - 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")
```

DT of s is the same as ST of s.

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								

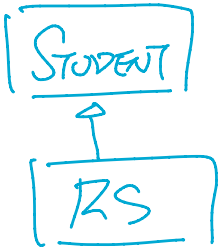
```
if (o instanceof C) {
    ...
}
```

```
if attached {C} o then
    ...
end
```

Polymorphism: Intuition

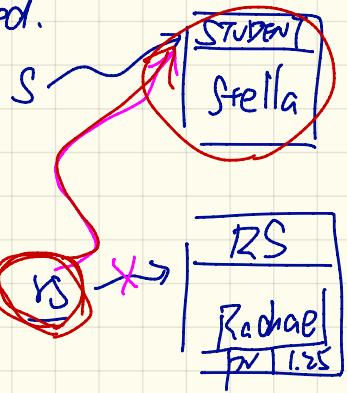
S := RS compiles.
substitute RS for S.

```
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? */
```



4. RS := S should not compile.

1. Assume RS := S compiled.



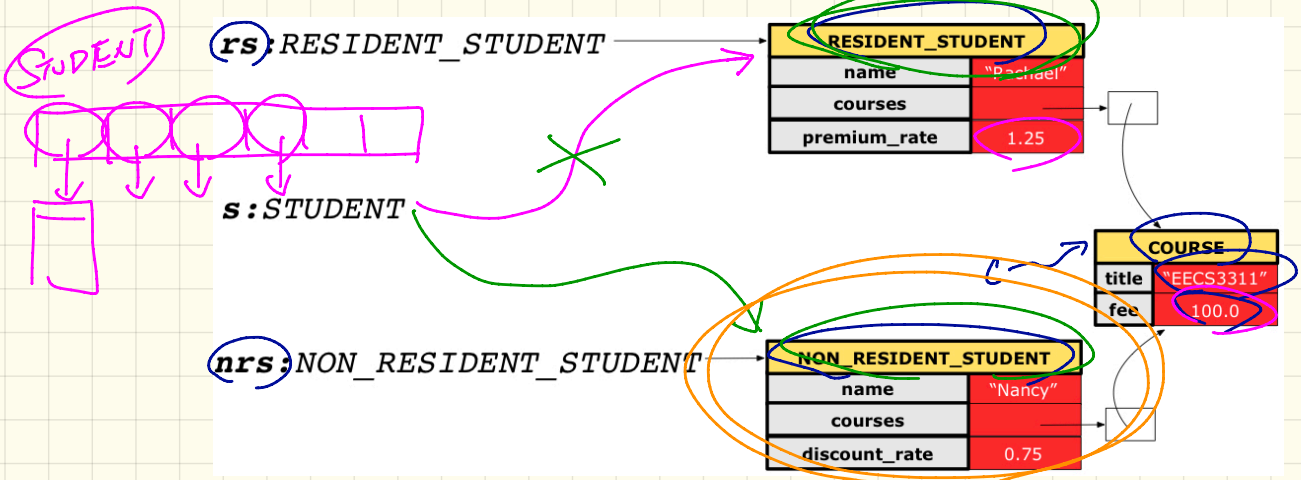
2. Expectations on RS? S.T.
name
counter
reg
tri.
pr
set_pr

3. ST: RS compile.
RS. pr
compiles : ST of RS declares pr

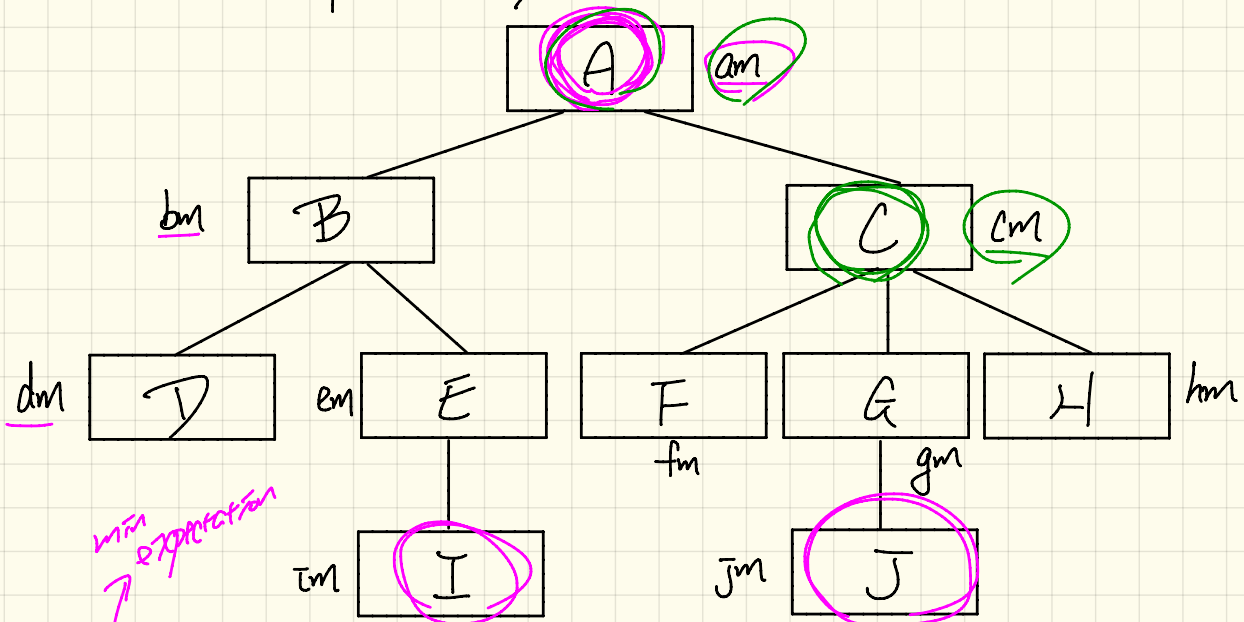
Runtime?
Crash : STUDENT object does not have pr.

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 = [redacted] end
```



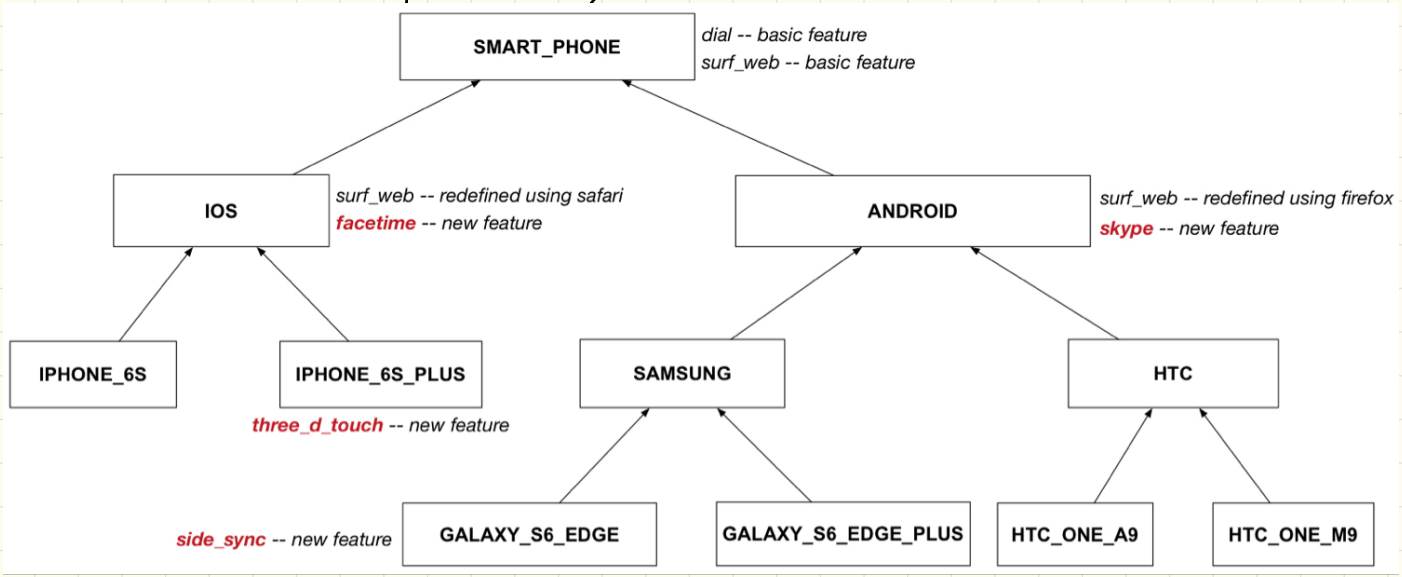
Inheritance Forms a Type Hierarchy (1)



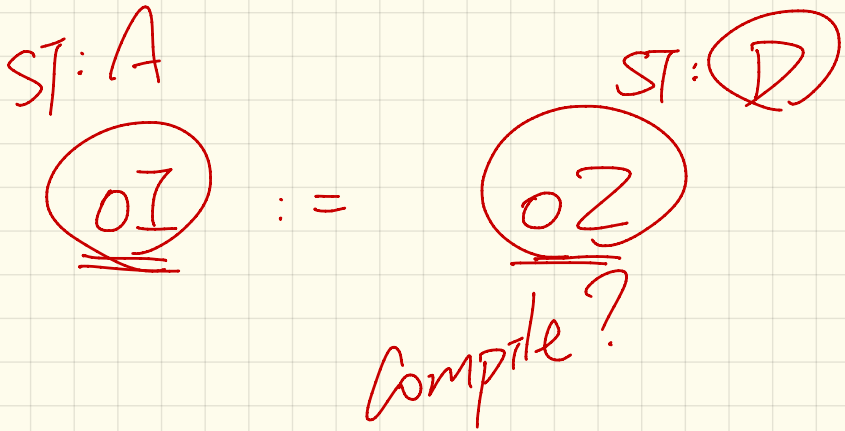
with expectation

	ancestors	expectations	descendants
A	A	am	all classes
C	A, C	am, cm	C, F, G, H, J
G			

Inheritance Forms a Type Hierarchy (2)

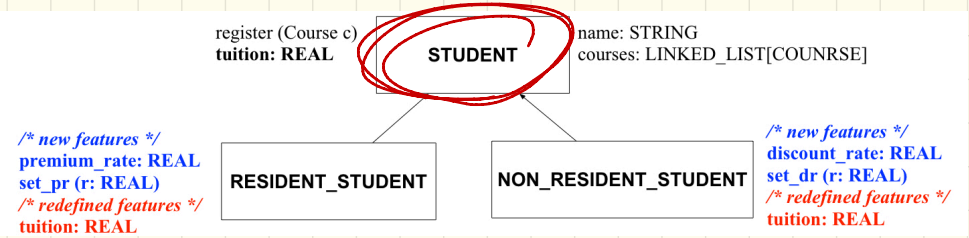


	ancestors	expectations	descendants
SMART_PHONE			
ANDROID			
GS6EP			

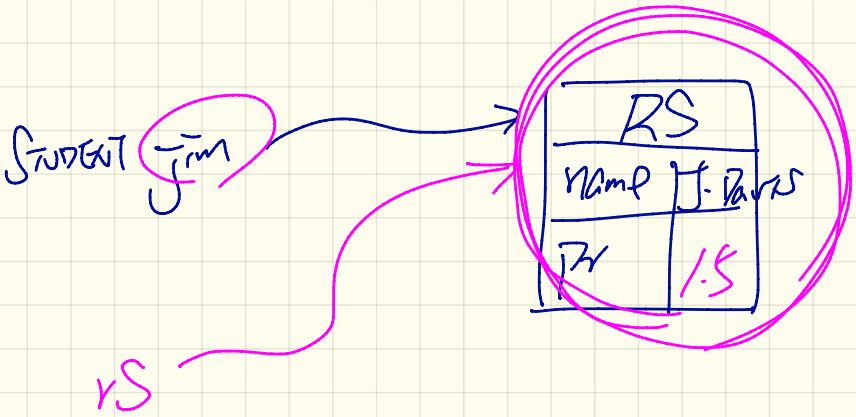


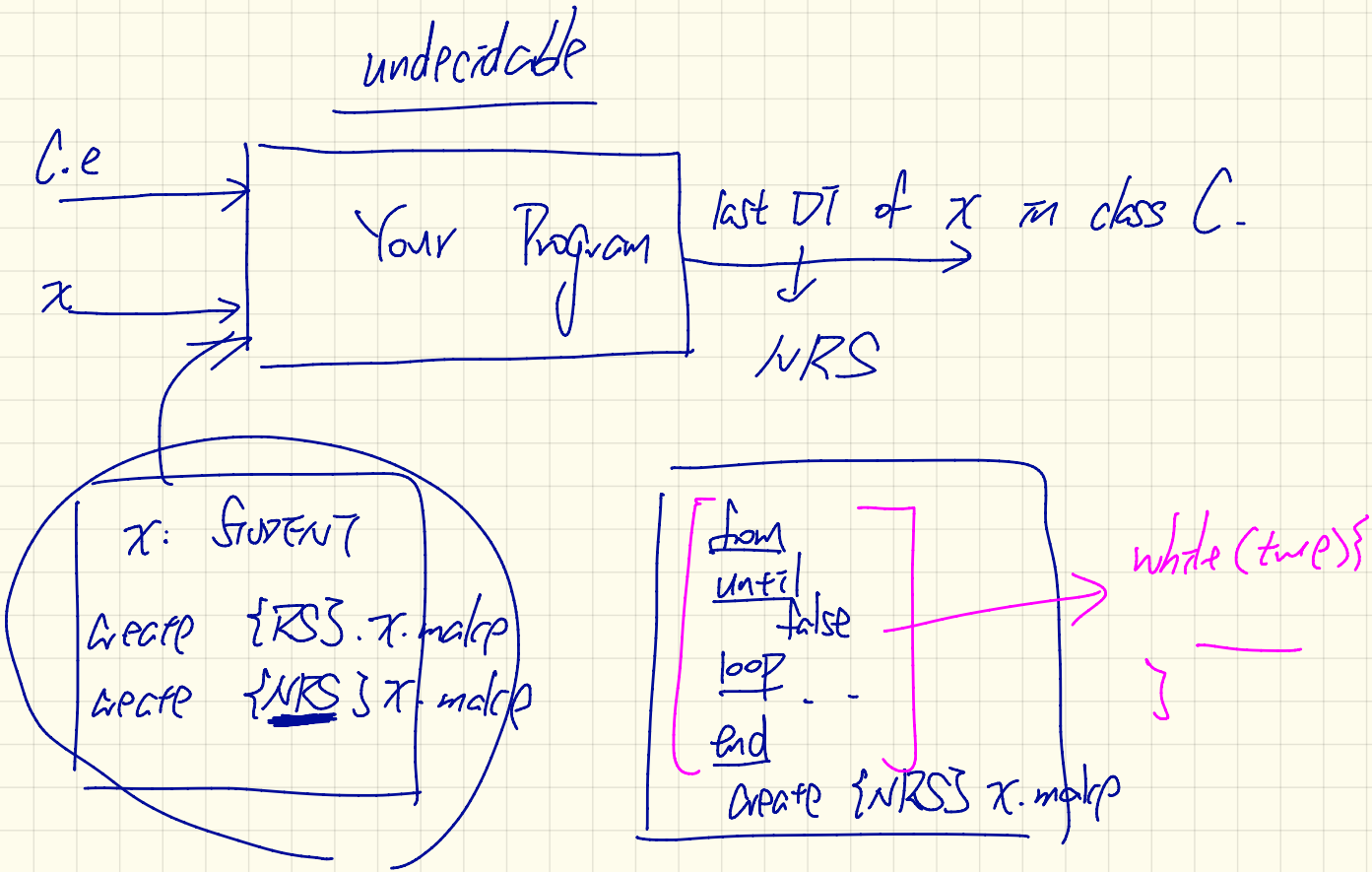
ST of o2 is a descendant of ST of o1.

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)
```

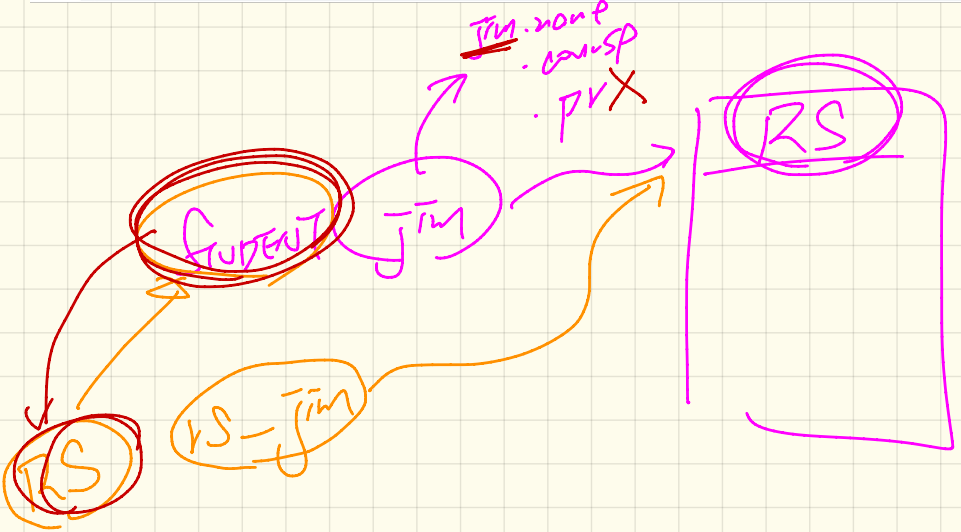



undecidable
C.e
x
Your Program
last DT of x in class C.
NRS
x: STUDENT
create {RSS}.x.make
create {NRS}x.make
from
until
false
loop
. . .
end
create {NRSS}x.make
while (true){
}

Type Cast : Syntax

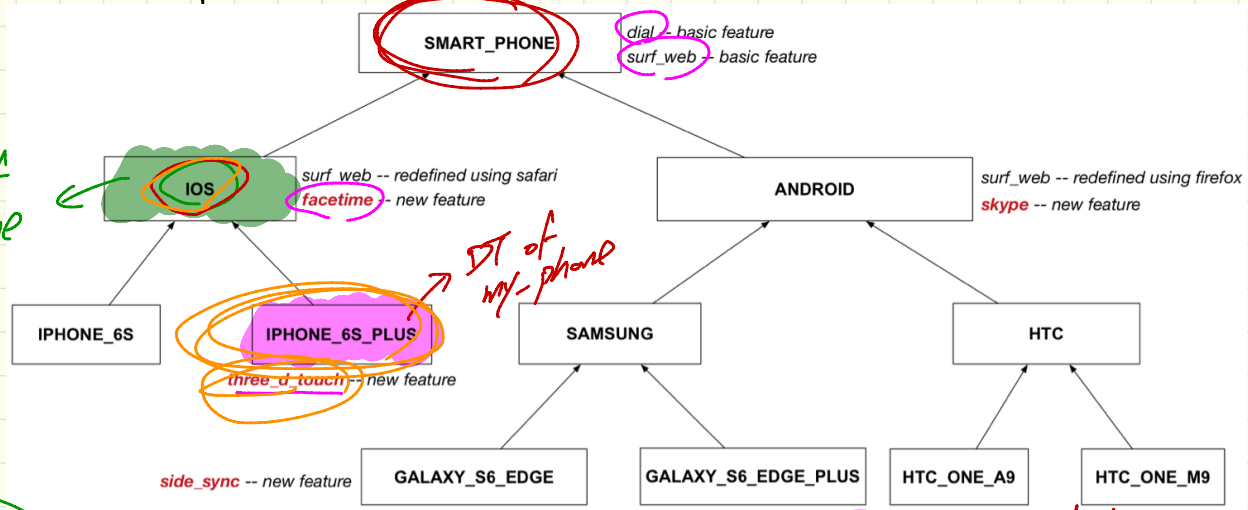
jim instanceof RS

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



cast
↳ upward cast
↳ restricting less expectations.
↳ downward cast
↳ allowing more expectations

Compilable Cast : Upward or Downward



ST of my_phone

ST of my_phone

```
my_phone: IOS
create {IPHONE_6S_PLUS} my_phone.make
  -- can only call features defined in IOS on myPhone
  -- dial, surf_web, facetime ✓  three_d_touch, skype ✗
check attached {SMART_PHONE} my_phone as sp then
  -- can now call features defined in SMART_PHONE on sp
  -- dial, surf_web ✓  facetime, three_d_touch, skype ×
end
check attached {IPHONE_6S_PLUS} my_phone as ip6s_plus then
  -- can now call features defined in IPHONE_6S_PLUS on ip6s_plus
  -- dial, surf_web, facetime, three_d_touch ✓  skype ×
end
```

upward

my_phone

d
s_w
facetime

sp. dial
s_w
ip6s_plus • three_d_touch

IOS my_phone

I_6S_P

sp

sp

I_6S_P ip6s_plus

Compilable Cast May Fail at Runtime

runtime cast error if the type is not an ancestor of the DT.

ST of mine

DT of mine

```
test_smart_phone_type_cast_violation
  local mine: ANDROID
  do create {SAMSUNG} mine.make
    -- ST of mine is ANDROID; DT of mine is SAMSUNG
    check attached {SMART_PHONE} mine as sp then ... end
    -- ST of sp is SMART_PHONE; DT of sp is SAMSUNG
    check attached {SAMSUNG} mine as samsung then ... end
    -- ST of samsung is SAMSNG; DT of samsung is SAMSUNG
    check attached {HTC} mine as htc then ... end
    -- Compiles ∵ HTC is descendant of mine's ST (ANDROID)
    -- Assertion violation
    -- ∵ HTC is not ancestor of mine's DT (SAMSUNG)
    check attached {GALAXY_S6_EDGE} mine as galaxy then ... end
    -- Compiles ∵ GALAXY_S6_EDGE is descendant of mine's ST (ANDROI
    -- Assertion violation
    -- ∵ GALAXY_S6_EDGE is not ancestor of mine's DT (SAMSUNG)
end
```

cast error

False

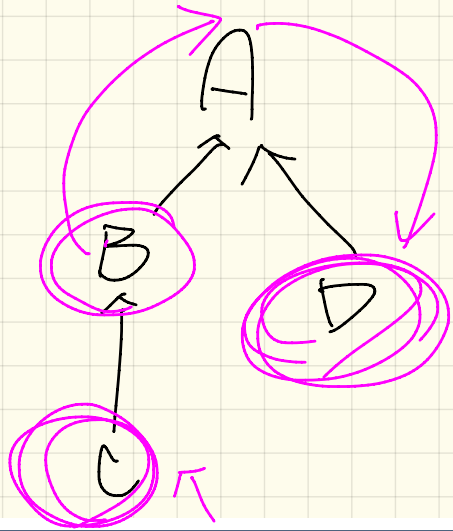
Assume the cast was ok.

galaxy → SAMSUNG

galaxy.side_sync

✓ compile

✗ runtime




```
1  local b: B ; d: D
2  do
3    create {C} b.make
4    check attached {D} b as temp then d := temp end
5  end
```

F

Compile ?