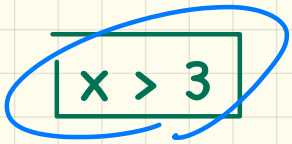
Lecture 10

Part 1

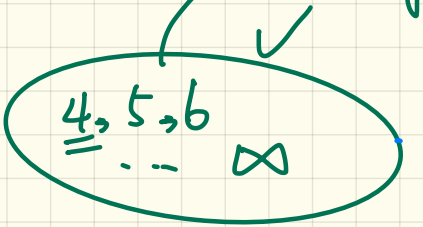
Contracts - Require Less vs. Ensure More

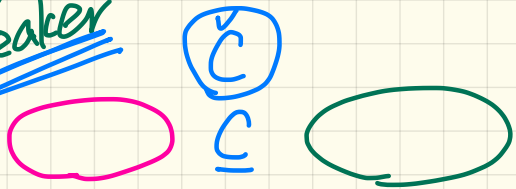
Assertions: Weak vs. Strong

Predicates → Boolean values
→ set of satisfying values



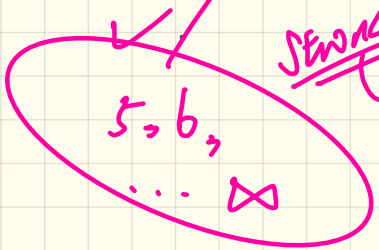
more accommodating, larger, weaker

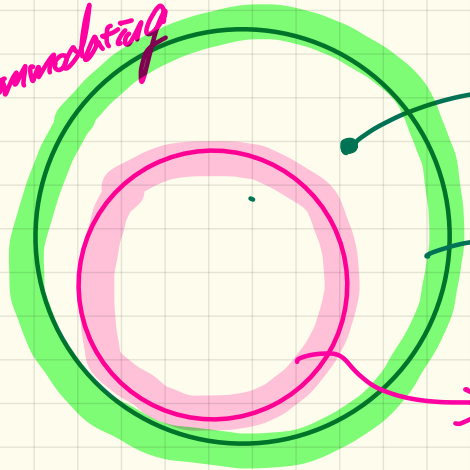




Weakest?
Strongest?

less accommodating, smaller, stronger



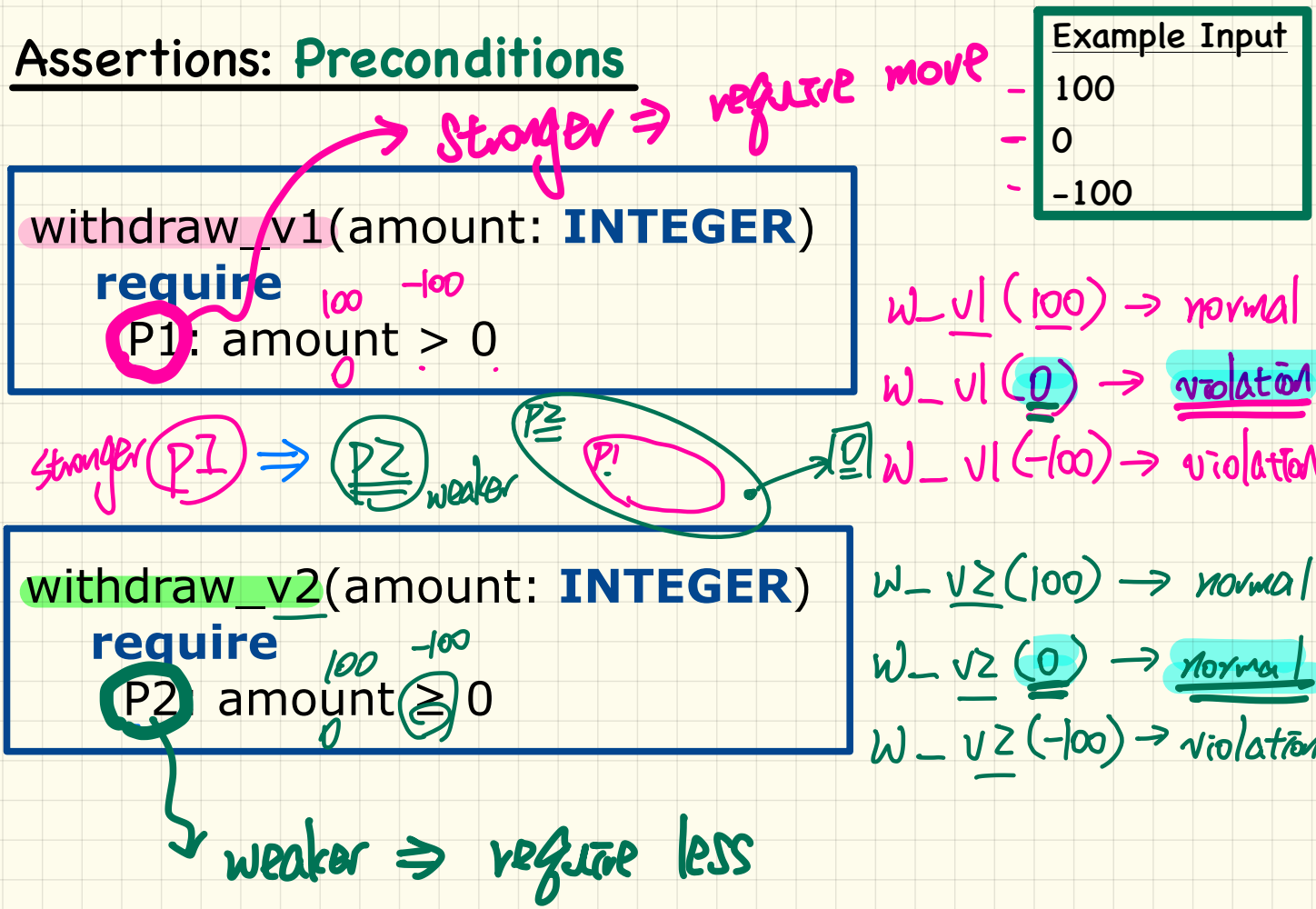


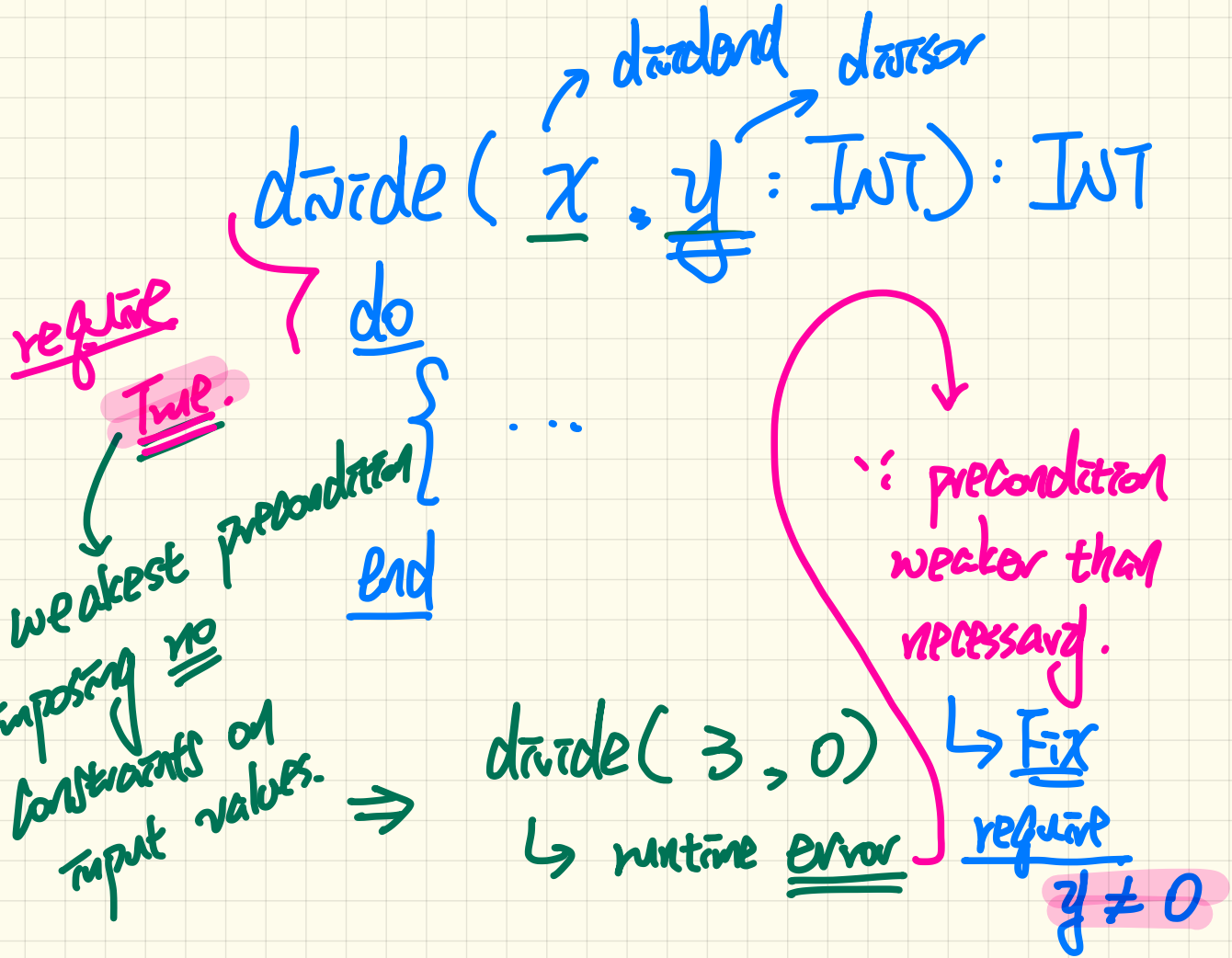
4

x > 3
⇝ weaker.

x > 4 ⇝
stronger.

Assertions: Preconditions




dividend
divisor
divide (x , y : INT) : INT
require
True.
do
...
end
weakest precondition
imposing no
constraints on
input values.
⇒
divide (3 , 0)
↳ runtime error
: precondition
weaker than
necessary.
↳ Fix
require
y ≠ 0

Assertions: Postconditions

$x \wedge y \Rightarrow x \vee y$

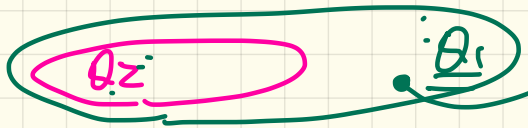
False → ϕ $\quad \phi \subseteq -$

```
f1(i: INTEGER): BOOLEAN
    ensure
      Q1: Result = (i > 0) or (i mod 2 = 0)
```

T T T

f1(79) → T
f1(-34) → T
f1(62) → T

Q2 ⇒ Q1



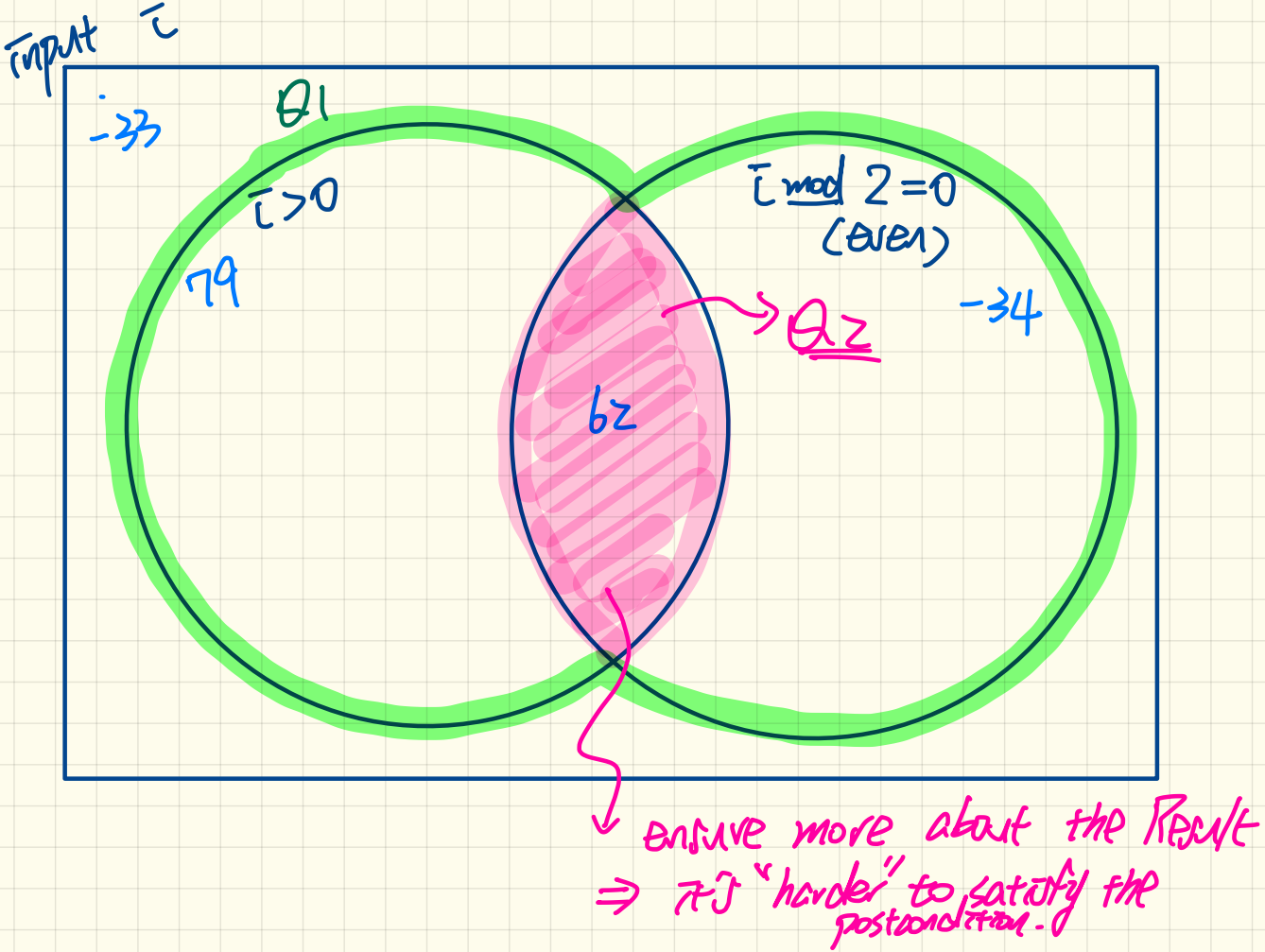

```
f2(i: INTEGER): BOOLEAN
    ensure
      Q2: Result = (i > 0) and (i mod 2 = 0)
```

✓ F F ✓

f2(79) → F
f2(-34) → F
f2(62) → T

Example Input
79
-34
62

What's the value of Result (output) that will satisfy the postcondition.

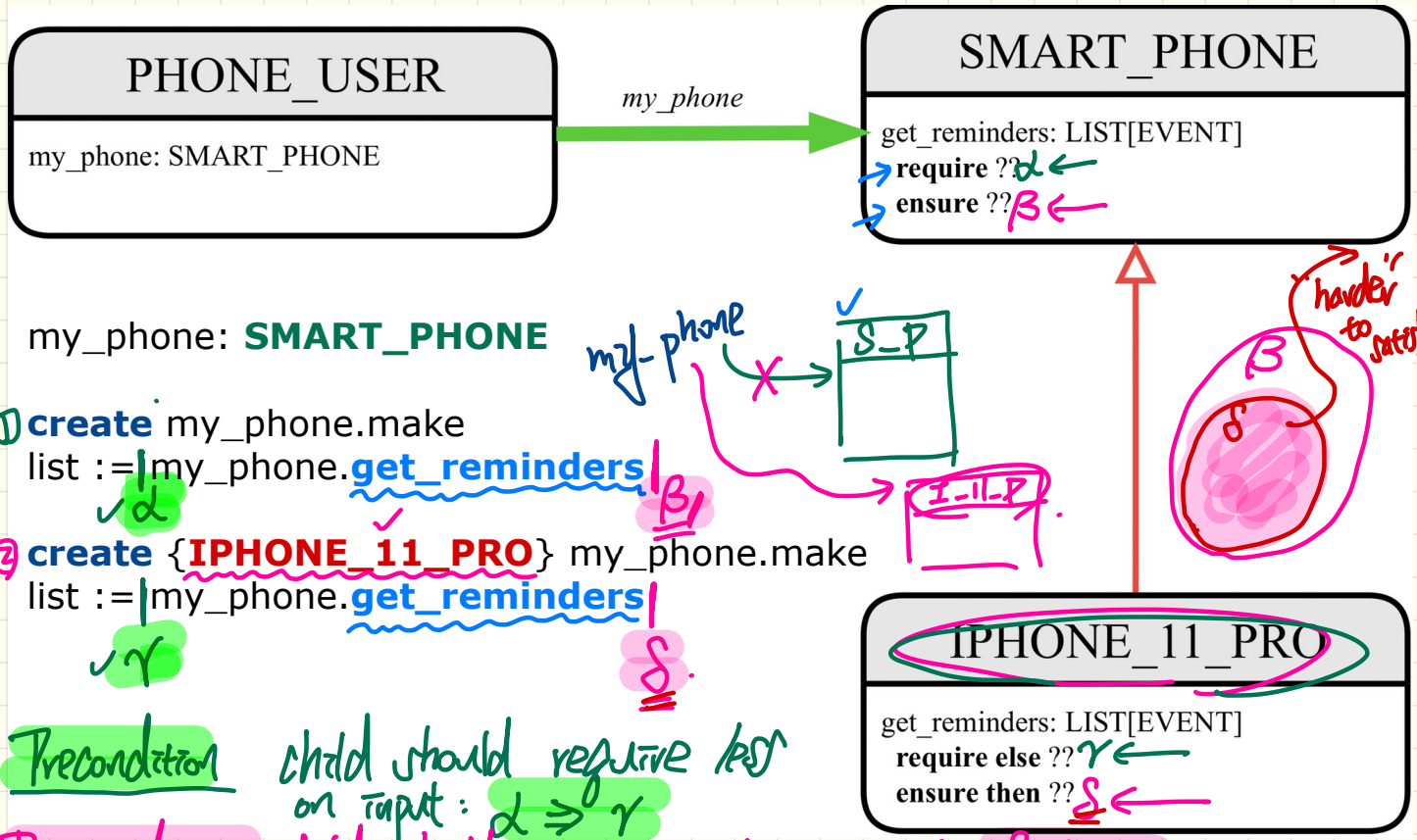

Input i
-33
Q1
i > 0
79
i mod 2 = 0
(even)
-34
Q2
Q2
ensure more about the Result
⇒ it's "harder" to satisfy the postcondition.

Lecture 10

Part 2

Inheritance & Contracts - Static Analysis

Subcontracting: Architectural View



Subcontracting: Example (1)

```
class SMART_PHONE
  get_reminders: LIST[EVENT]
    require
      α: battery_level ≥ 0.1 -- 10%
    ensure
      β: ∀e:Result | e happens today
end
```

```
class IPHONE_11_PRO
inherit SMART_PHONE redefine get_reminders end
  get_reminders: LIST[EVENT]
    require else
      γ: battery_level ≥ 0.15 -- 15%
    ensure then
      δ: ∀e:Result | e happens today or tomorrow
end
```

poor design :: stronger precondition

$\alpha \Rightarrow \beta$

$bl \geq 10\% \Rightarrow bl \geq 15\%$

α γ

10, 11, 12, 13, 14, 15, 16 ... γ

C.E.

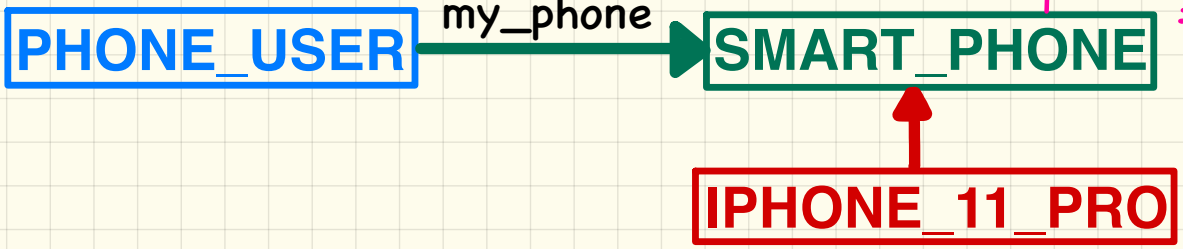
10, 11, 12, 13, 14 satisfy α but not γ

```
my_phone: SMART_PHONE

create my_phone.make
list := my_phone.get_reminders

create {IPHONE_11_PRO} mine.make
list := my_phone.get_reminders
```

α ✓ 12%.

γ 12%.

↳ precondition violation ⇒ shock to user.



Subcontracting: Example (2)

```
class SMART_PHONE
  get_reminders: LIST[EVENT]
    require
      α: battery_level ≥ 0.1 -- 10%
    ensure
      β: ∀e: Result | e happens today
end
```

```
class IPHONE_11_PRO
inherit SMART_PHONE redefine get_reminders end
  get_reminders: LIST[EVENT]
    require else
      γ: battery_level ≥ 0.15 -- 15%
    ensure then
      δ: ∀e: Result | e happens today or tomorrow
end
```

↳ poorly designed ⁉ ensures less

```
my_phone: SMART_PHONE

create my_phone.make
list := my_phone.get_reminders

create {IPHONE_11_PRO} mine.make
list := my_phone.get_reminders
```

Rule: $\delta \Rightarrow \beta$

δ: e happens today ∨ tmw

$\Rightarrow$?

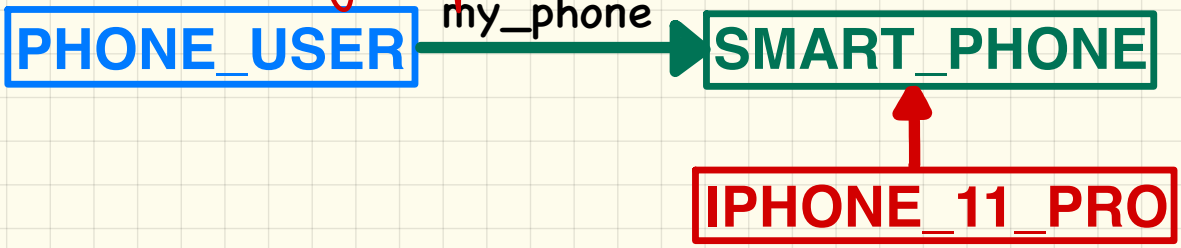
β: e happens today

Imp. output tmw. e. with δ

tomor. β today

δ.

tmw events only.



Lecture 10

Part 3

Inheritance & Contracts - Runtime Checks

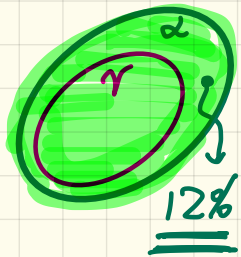
Subcontracting: Example (1)

```
class SMART_PHONE
  get_reminders: LIST[EVENT]
    require
      α: battery_level ≥ 0.1 -- 10%
    ensure
      β: ∀e : Result | e happens today
end
```

```
class IPHONE_11_PRO
inherit SMART_PHONE redefine get_reminders end
  get_reminders: LIST[EVENT]
    require else
      γ: battery_level ≥ 0.15 -- 15%
    ensure then
      δ: ∀e : Result | e happens today or tomorrow
end
```

Invalid

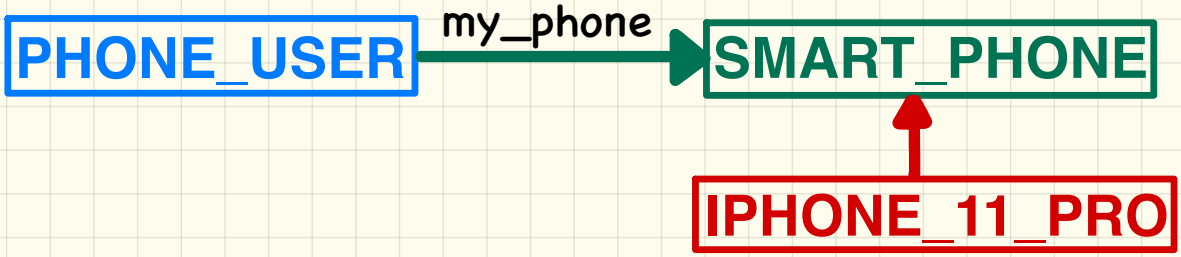
$\because \gamma \Rightarrow \alpha$



Runtime :

$\neg\alpha \vee \gamma \equiv \alpha$ (T)

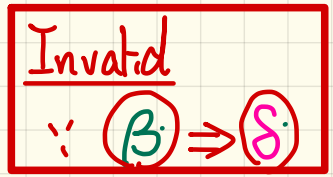
```
my_phone: SMART_PHONE

create my_phone.make
list := my_phone.get_reminders

create {IPHONE_11_PRO} mine.make
list := my_phone.get_reminders
```



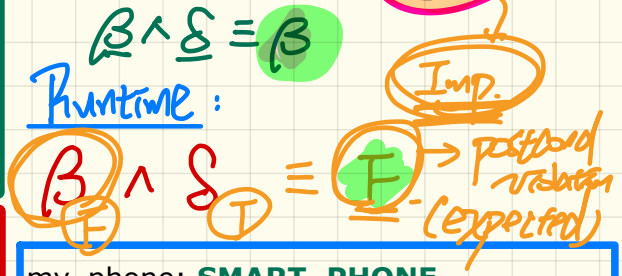
Subcontracting: Example (2)

```
class SMART_PHONE
  get_reminders: LIST[EVENT]
    require
      α: battery_level ≥ 0.1 -- 10%
    ensure
      β: ∀e : Result | e happens today
end
```

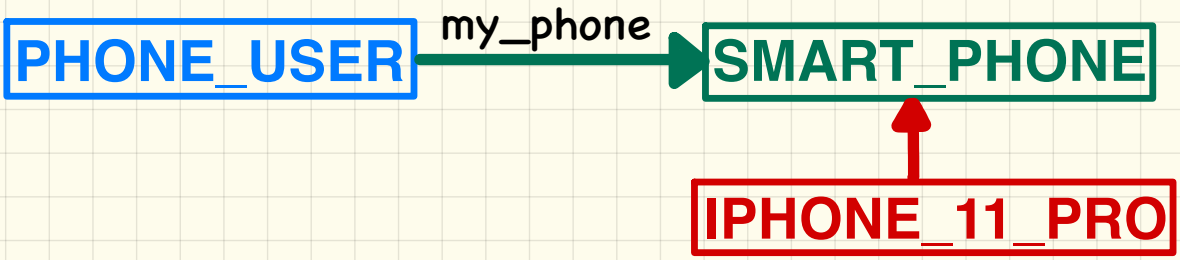
```
class IPHONE_11_PRO
inherit SMART_PHONE redefine get_reminders end
  get_reminders: LIST[EVENT]
    require else
      γ: battery_level ≥ 0.15 -- 15%
    ensure then
      δ: ∀e : Result | e happens today or tomorrow
end
```



$\beta \wedge \delta \equiv \beta$

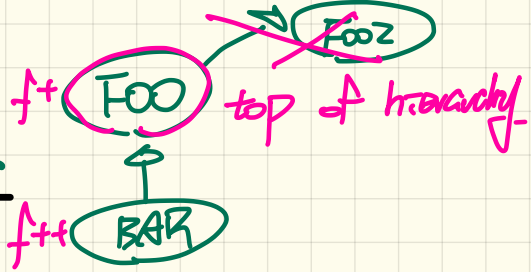



```
my_phone: SMART_PHONE

create my_phone.make
list := my_phone.get_reminders

create {IPHONE_11_PRO} mine.make
list := my_phone.get_reminders
```



Contract Re-Declaration:

Missing Pre-Condition in Ancestor




```
class FOO
  f
    do ...
    end
end
```

```
class BAR
inherit FOO redefine f end
  f require else new_pre
    do ...
    end
end
```

require True

True V new_pre

≡ True → at runtime, even if DT ⇒ BAR only check True to proceed

Contract Re-Declaration:

Missing Post-Condition in Ancestor

```
class FOO
  f
    do ...
    end
end
```

```
class BAR
inherit FOO redefine f end
  f
    do ...
    ensure then new_post
    end
end
```

Stronger

ENSURE True.

Runtime.

True ∧ new_post.

≡ new_post.

When DT in BAR check new_post for postcond.

Contract Re-Declaration:

Missing Pre-Condition in Descendant

```
class FOO
  f require
      original_pre
    do ...
    end
end
```

```
class BAR
inherit FOO redefine f end
  f
    do ...
    end
end
```

require else

~~or~~ False.

original_pre ∨ ?? False

≡ original_pre

Contract Re-Declaration:

Missing Post-Condition in Descendant

```
class FOO
  f
    do ...
    ensure
      original_post
    end
end
```

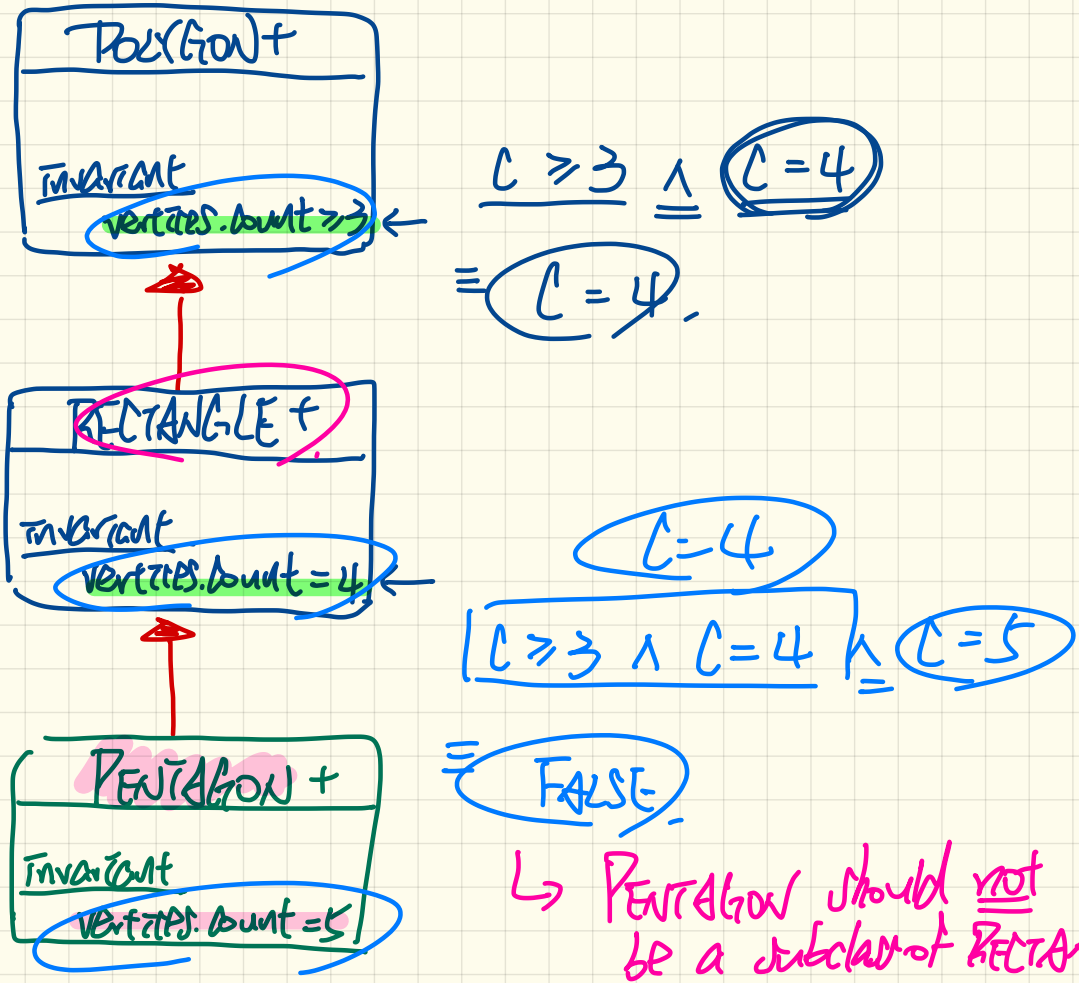
```
class BAR
inherit FOO redefine f end
  f
    do ...
  end
end
```

ensure then

~~??~~ True

original_post ∧ ?? True

≡ original_post

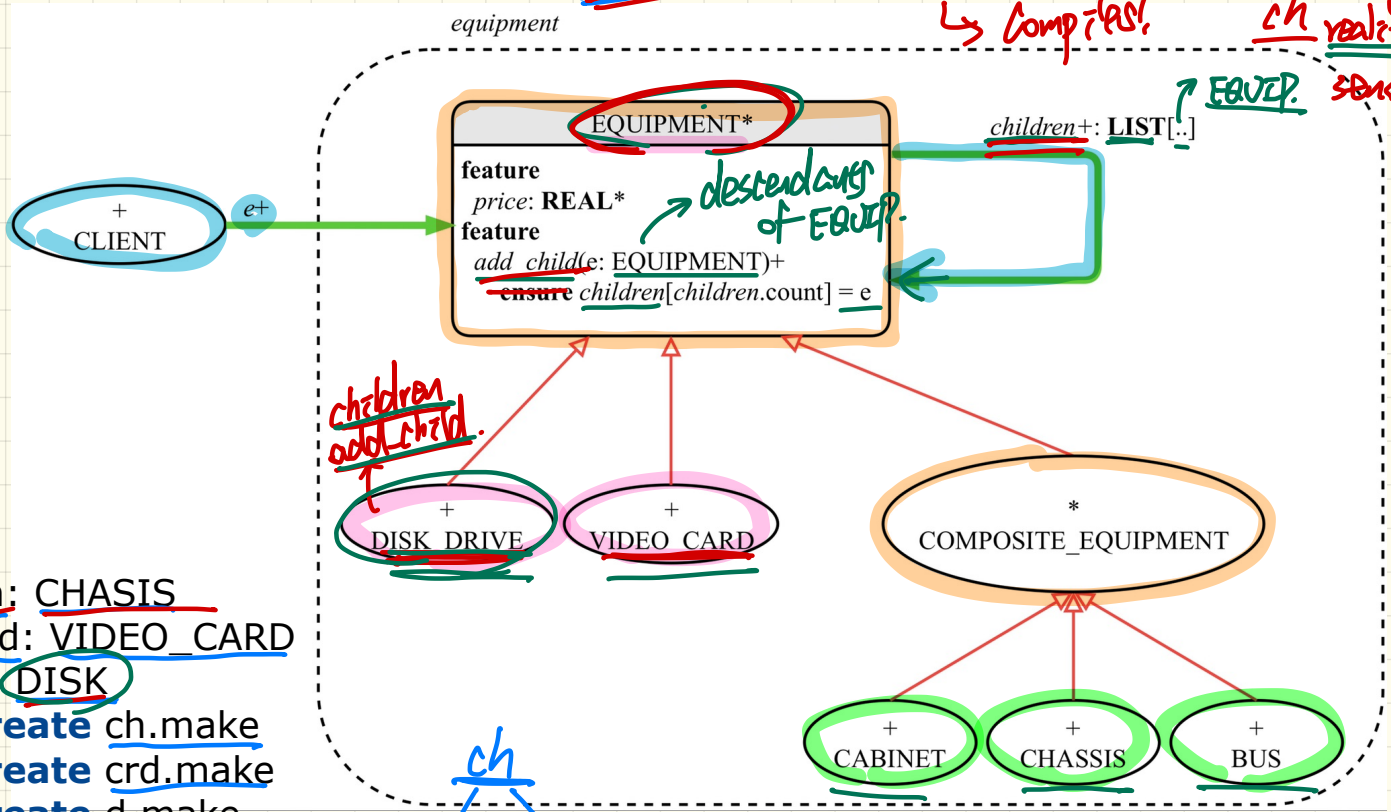

POLYGON+
INVARIANT
vertices.count >= 3
RECTANGLE+
INVARIANT
vertices.count = 4
PENTAGON+
INVARIANT
vertices.count = 5
C >= 3 ∧ C = 4
C = 4
C = 4
C >= 3 ∧ C = 4 ∧ C = 5
FALSE
↳ PENTAGON should not be a subclass of RECTAN.

Lecture 10

Part 4

Recursive Systems - Design Attempts

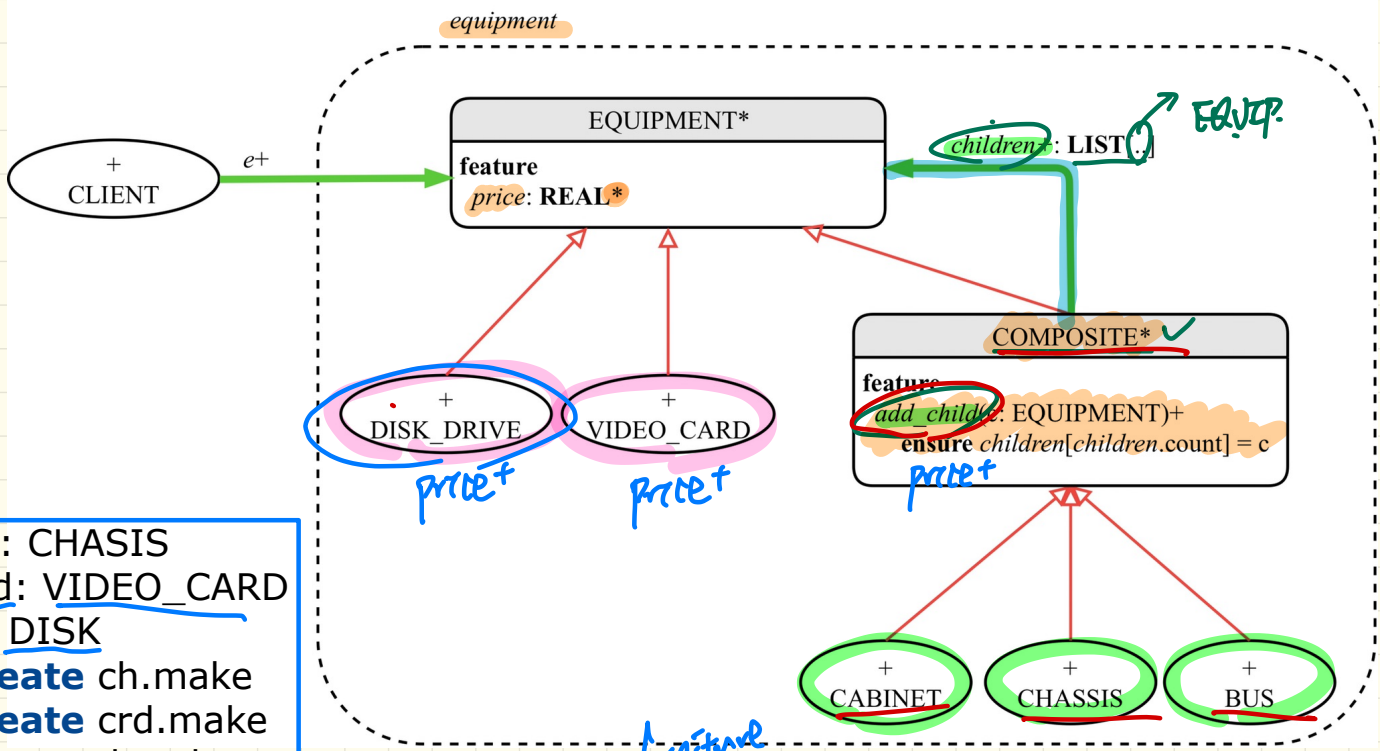
First Design Attempt




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

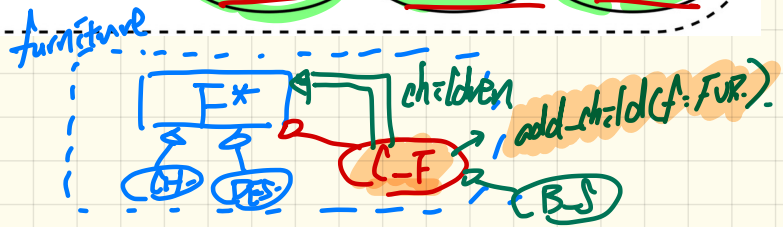
Second Design Attempt

SCP.




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

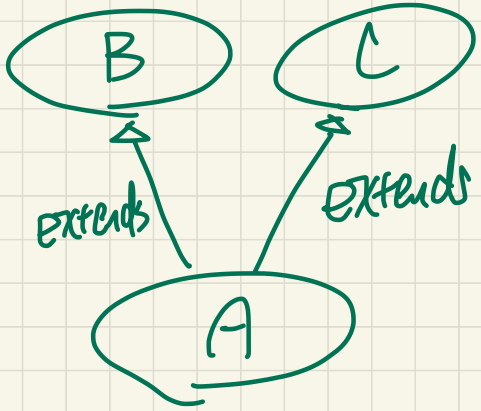
→ Compile? not.



Lecture 10

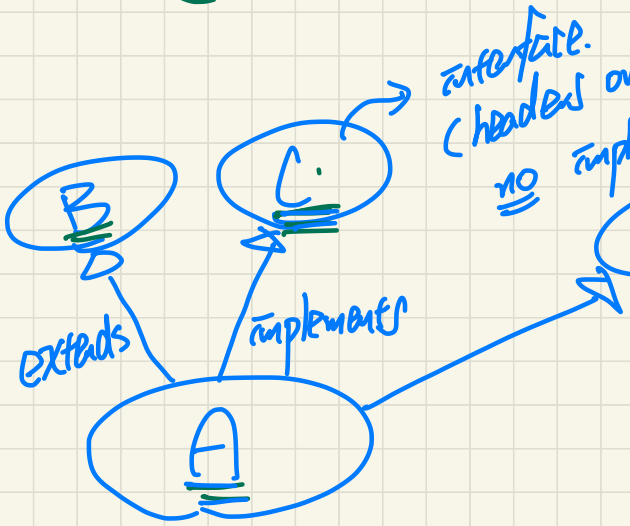
Part 5

Multiple Inheritance



class A extends B, C ✗

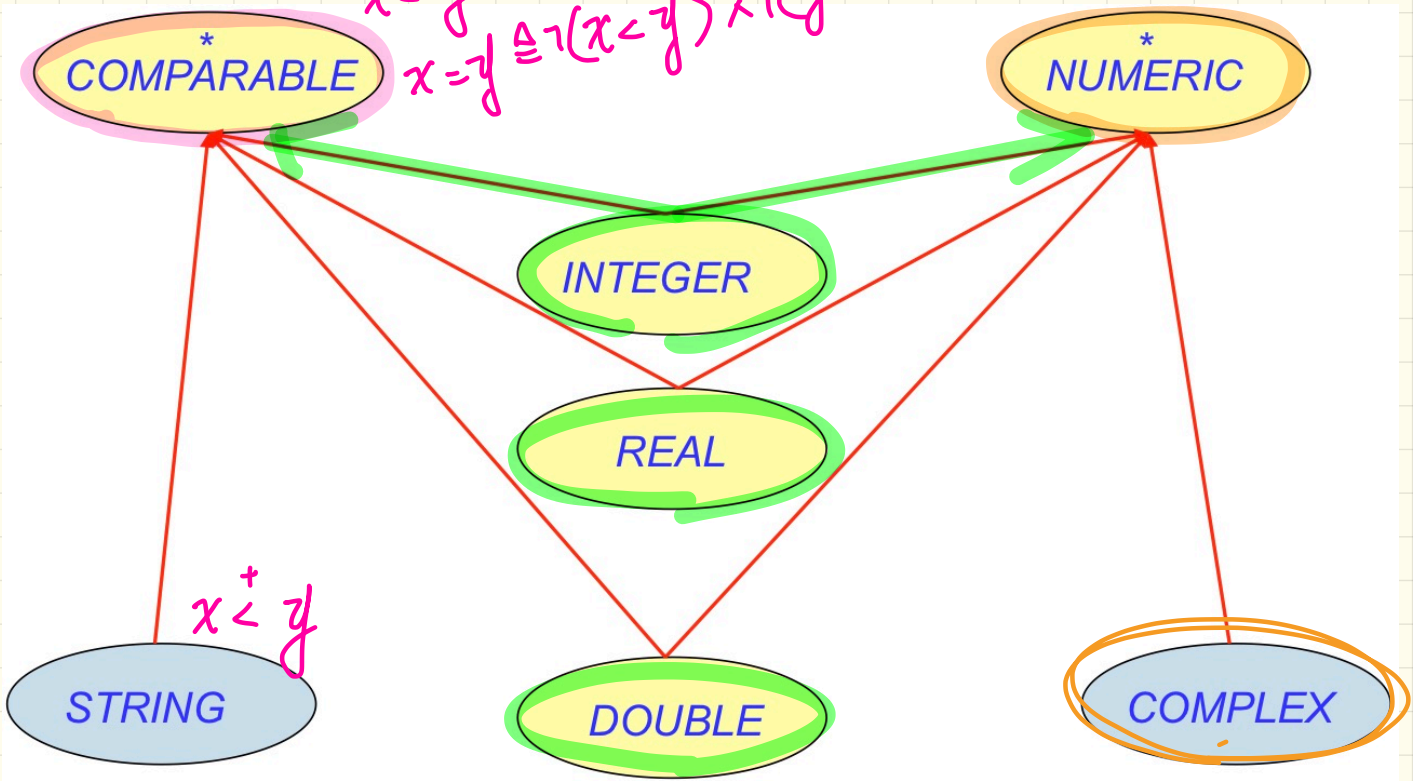
M.I. only supported in Java by a very limited way



class A extends B ✓.

implements C, D

Multiple Inheritance: Example



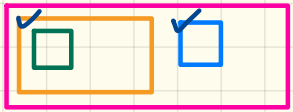
Multiple Inheritance: Exercise

```
class RECTANGLE
  feature -- Queries
    width, height: REAL
    xpos, ypos: REAL
  feature -- Commands
    make (w, h: REAL)
    change_width
    change_height
    move
end
```

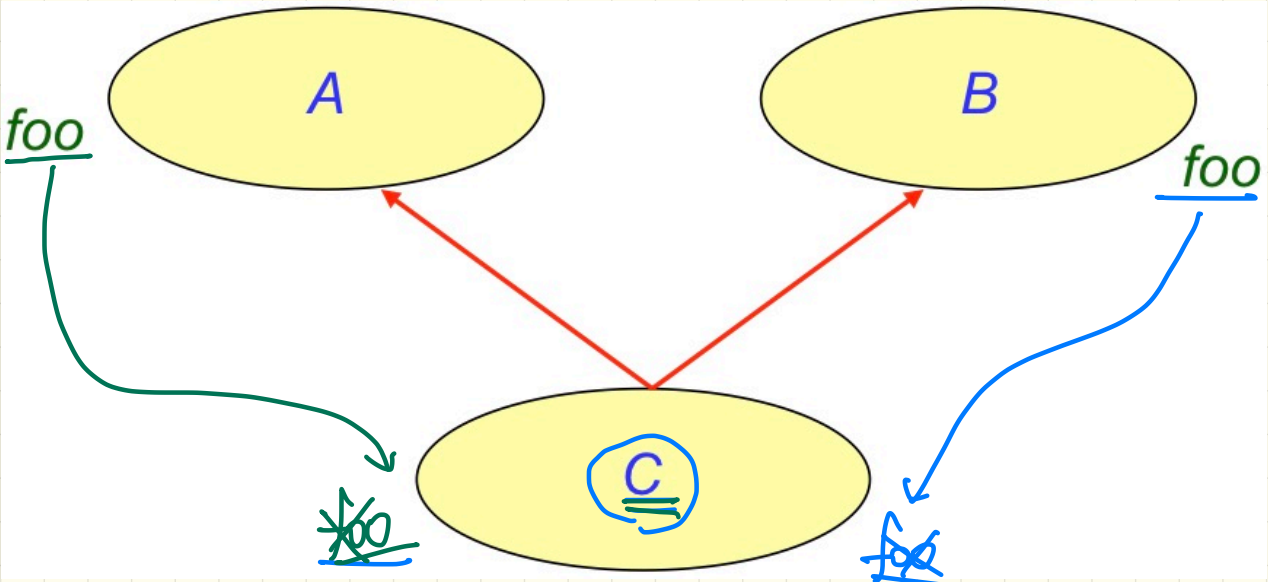
```
class TREE[G]
  feature -- Queries
    descendants: ITERABLE[G]
  feature -- Commands
    add (c: G)
        -- Add a child 'c'.
end
```

```
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
```

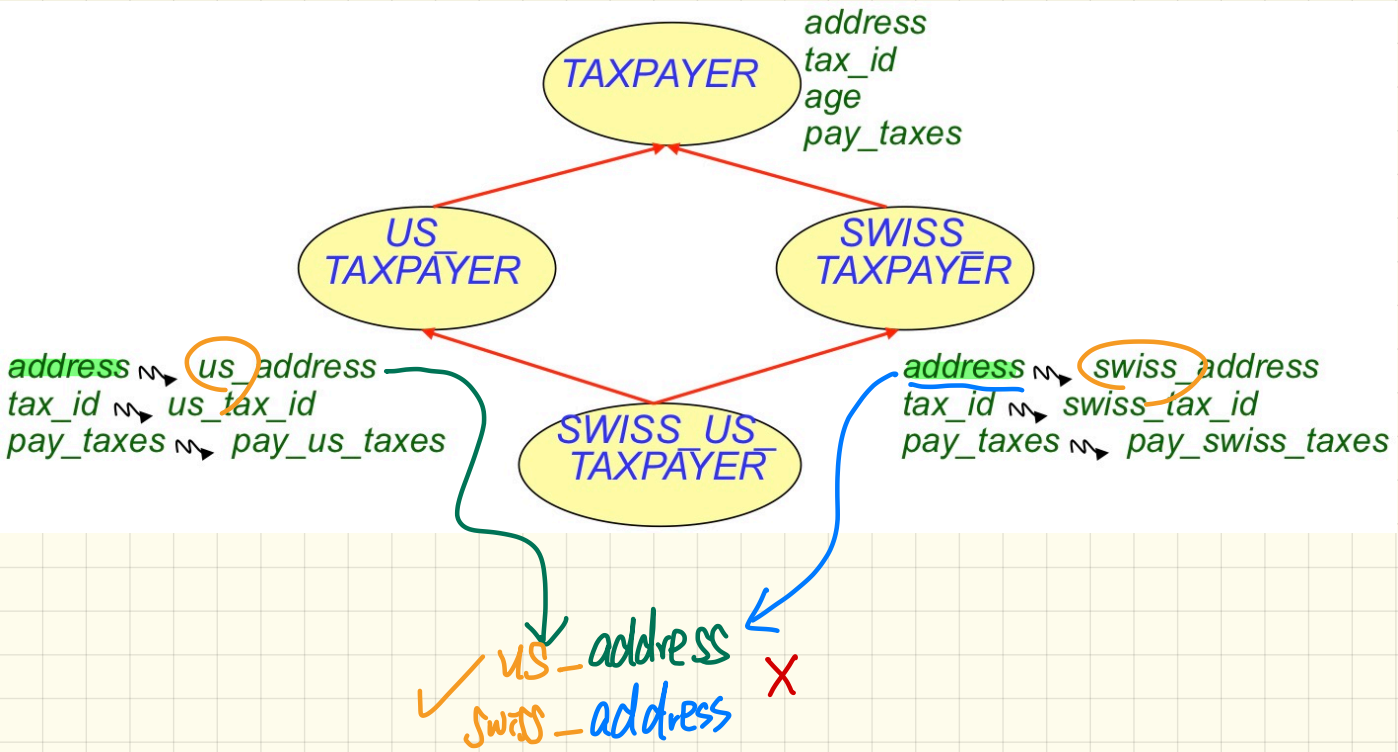


Multiple Inheritance: Name Clashes



x : C → exp: foo, Zoo.

Multiple Inheritance: Name Clashes

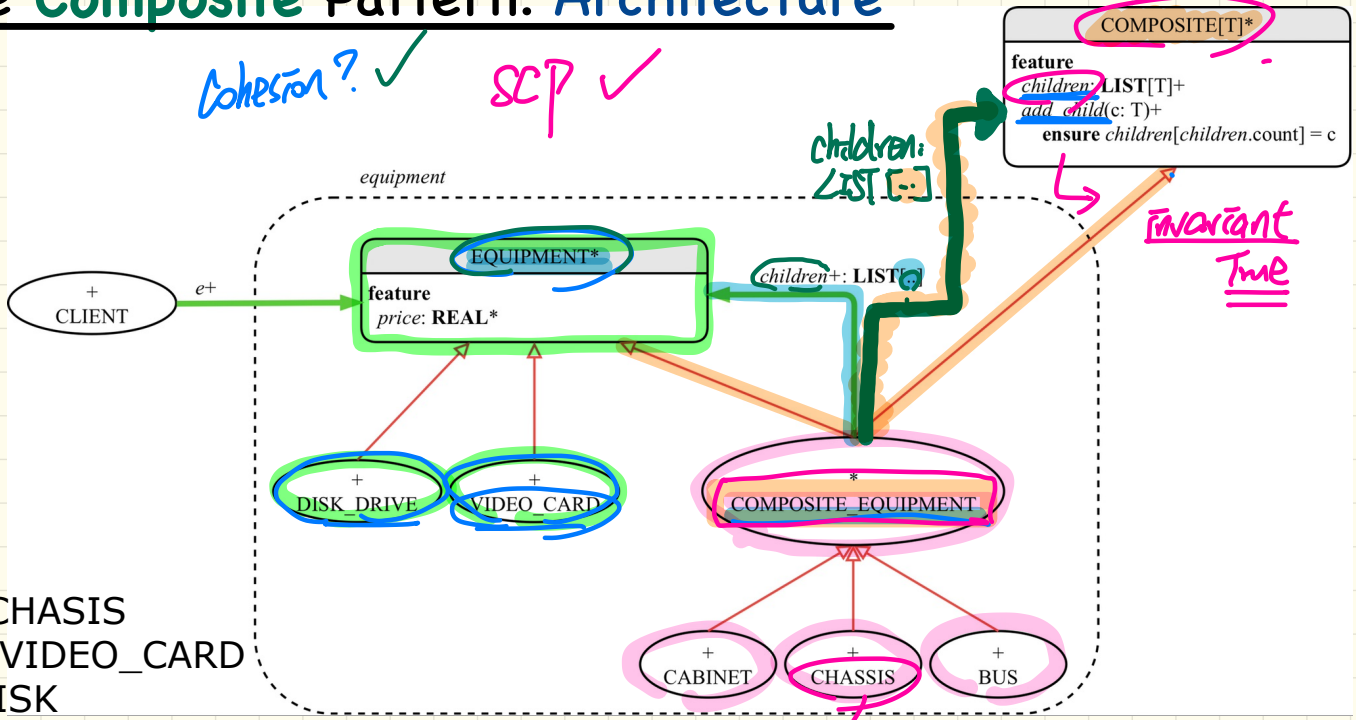


Lecture 10

Part 6

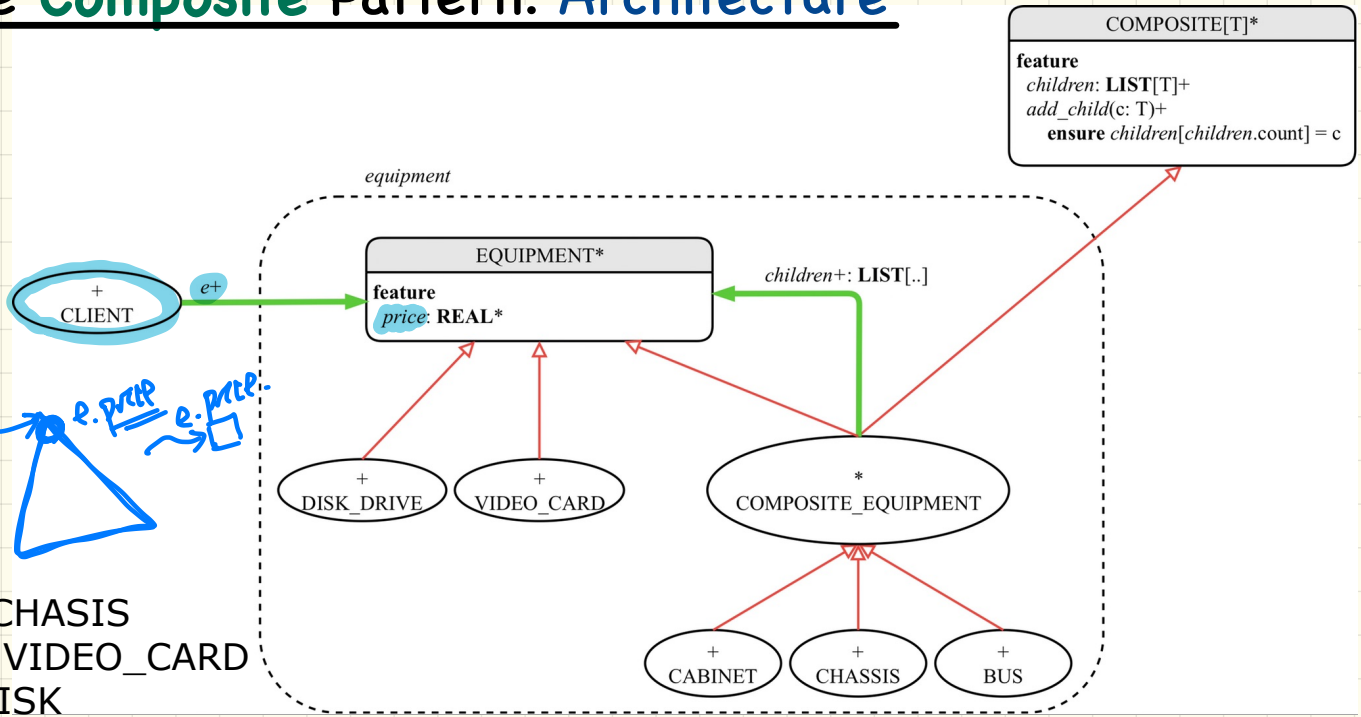
Composite Design Pattern

The Composite Pattern: Architecture



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

The Composite Pattern: Architecture



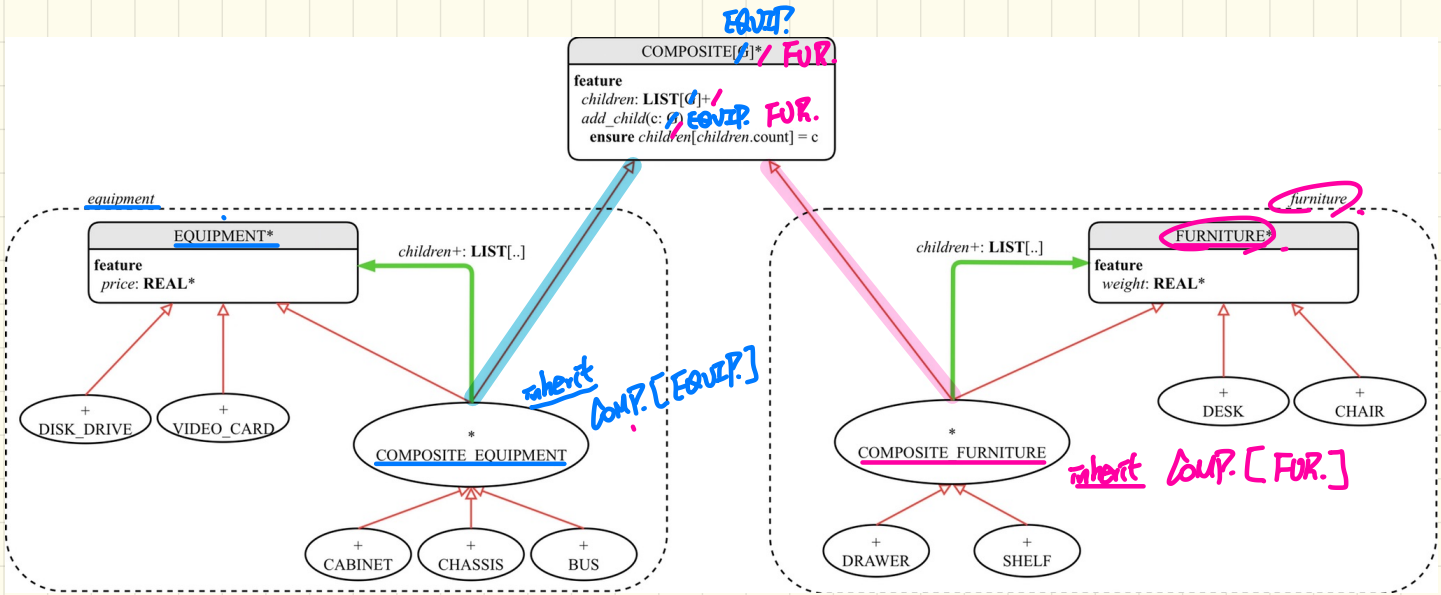
ch: CHASIS
crd: VIDEO_CARD
d: DISK
create ch.make
create crd.make
create d.make
ch.add_child(crd)
ch.add_child(d)
crd.add_child(d)

Why is COMPOSITE a separate class?

↳ satisfies SCP.

The Composite Pattern: Architecture

COMPOSITE class is **reusable** by instances of the composite pattern.



The Composite Pattern: Implementation

```
deferred class
  EQUIPMENT
feature
  name: STRING
  price: REAL deferred end
end
```

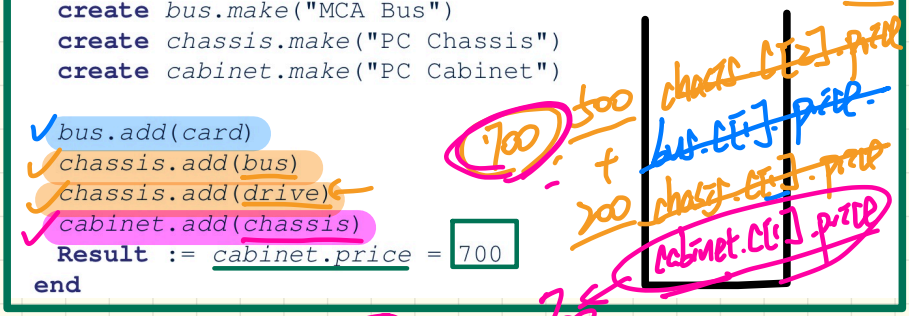
```
deferred class
  COMPOSITE[T]
feature
  children: LINKED_LIST[T]

  add_child (c: T)
    do
      children.extend (c) -- Polymorphism
    end
end
```

```
class
  CARD
inherit
  EQUIPMENT
feature {NONE}
  unit_price: REAL
feature
  make (n: STRING; p: REAL)
    do name := n ; unit_price := p end
  price
    do Result := unit_price end
end
```

```
class
  COMPOSITE_EQUIPMENT
inherit
  EQUIPMENT
  COMPOSITE [EQUIPMENT]
create
  make
feature
  make (n: STRING)
    do name := n ; create children.make end
  price : REAL -- price is a query
      -- Sum the net prices of all sub-equipments
    do
      across
        children is c
      loop
        Result := Result + c.price -- dynamic binding
      end
    end
end
```

Testing the Composite Pattern

```
test_composite_equipment: BOOLEAN
  local
    card, drive: EQUIPMENT
    cabinet: CABINET -- holds a CHASSIS
    chassis: CHASSIS -- contains a BUS and a DISK_DRIVE
    bus: BUS -- holds a CARD
  do
    create {CARD} card.make("16Mbs Token Ring", 200)
    create {DISK_DRIVE} drive.make("500 GB harddrive", 500)
    create bus.make("MCA Bus")
    create chassis.make("PC Chassis")
    create cabinet.make("PC Cabinet")

    bus.add(card)
    chassis.add(bus)
    chassis.add(drive)
    cabinet.add(chassis)
    Result := cabinet.price = 700
  end
```




```
class
  CARD
inherit
  EQUIPMENT
feature {NONE}
  unit_price: REAL
feature
  make (n: STRING; p: REAL)
    do name := n ; unit_price := p end
  price
    do Result := unit_price end
end
```

```
class
  COMPOSITE_EQUIPMENT
inherit
  EQUIPMENT
  COMPOSITE [EQUIPMENT]
create
  make
feature
  price : REAL -- price is a quer
      -- Sum the net prices of all
    do
      across
        children is c
      loop
        Result := Result + c.price
      end
    end
end
```