EECS3311 Software Design (Fall 2020)

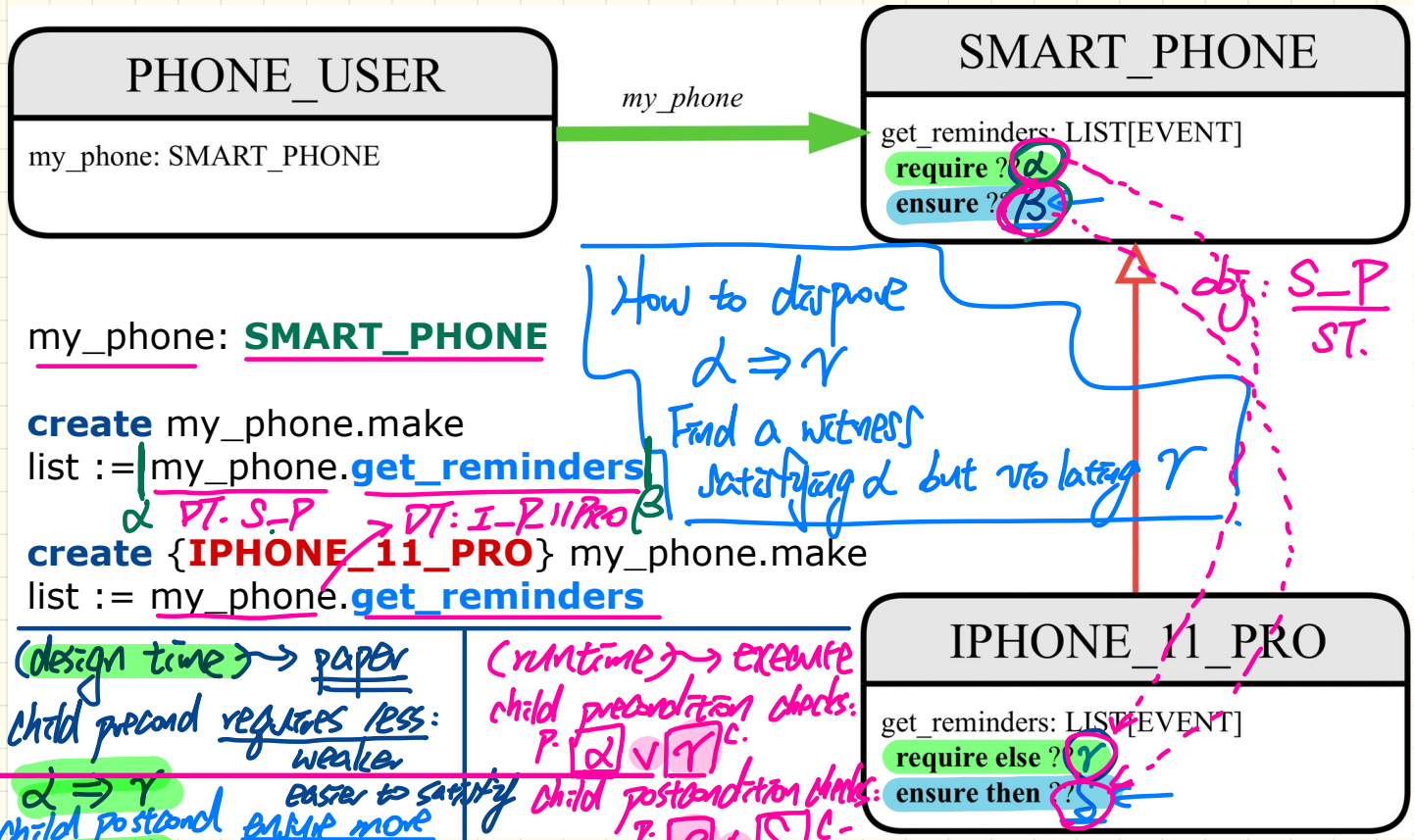
Q&A - Lecture Series W10

Monday, November 23

Subcontracting: Architectural View

$P \Rightarrow Q$

harder to satisfy?



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
```

satisfies

12%

```
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
```

violates

stronger ⇒ requires more

⇒ poor design.

The compiler does not prove or disprove:

$$\alpha \Rightarrow \gamma$$

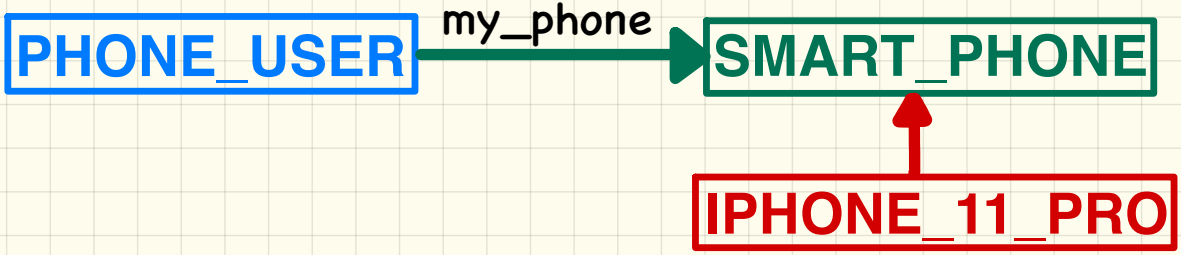
At runtime:

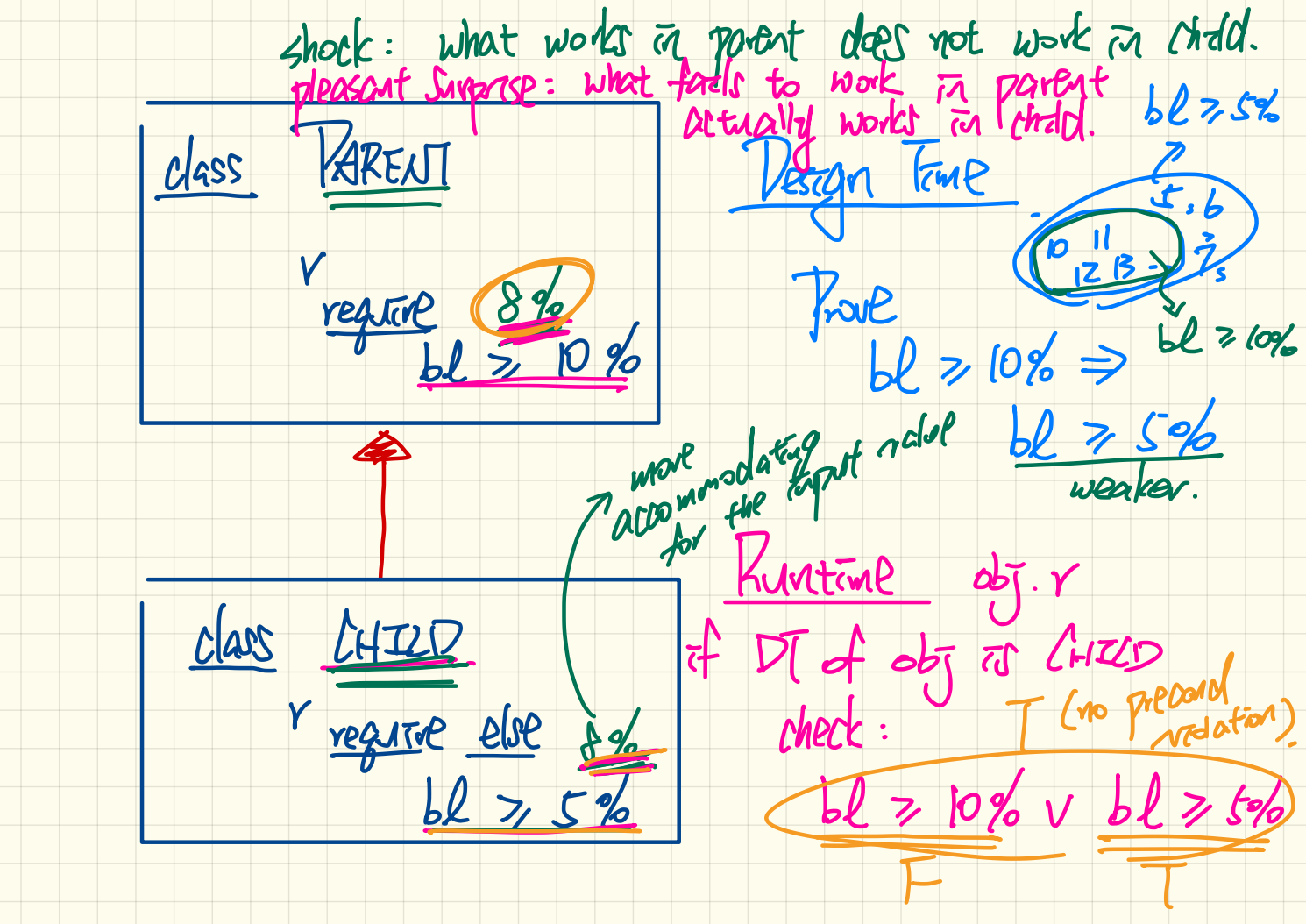
$$T \; \alpha \; V \; \gamma \; F \equiv T$$

```
my_phone: SMART_PHONE

create my_phone.make
list := my_phone.get_reminders

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




shock: what works in parent does not work in child.
pleasant surprise: what fails to work in parent actually works in child.
class PARENT
r
require
8%
bl ≥ 10%
class CHILD
r
require else
5%
bl ≥ 5%
more accommodating for the input value
Design Time
Prove
bl ≥ 10% ⇒
bl ≥ 5%
weaker.
bl ≥ 5%
a, b
10 11 12 13
bl ≥ 10%
Runtime obj.r
if DT of obj is CHILD
check:
T (no precond violation)
bl ≥ 10% ∨ bl ≥ 5%
F
T

Subcontracting: Example (2)

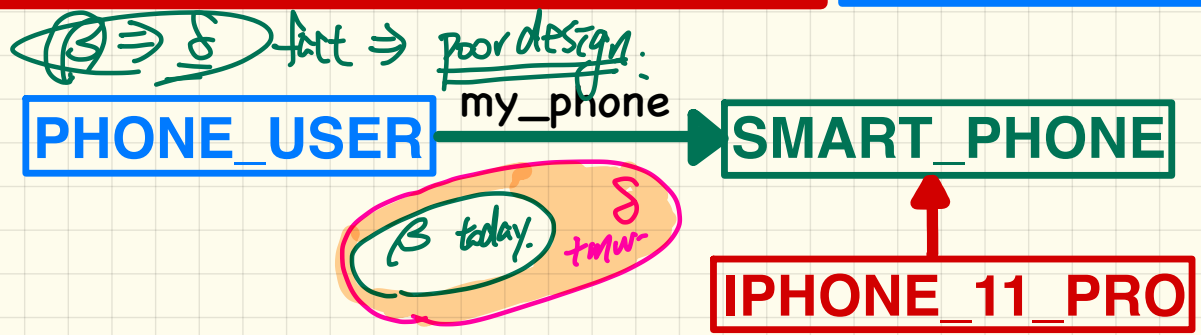
```
class SMART_PHONE
  get_reminders: LIST[EVENT]
    require
      α: battery_level ≥ 0.1 -- 10%
    ensure
      β: ∀e: Result | e happens today
end
```

```
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
```

```
my_phone: SMART_PHONE

create my_phone.make
list := my_phone.get_reminders

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



How is a Poor Design Checked at Runtime?

```
class FOO
  f
    require
      original_pre
    ensure
      original_post
    end
end
```

α

β

```
class BAR
inherit FOO redefine f end
  f
    require else
      new_pre
    ensure then
      new_post
    end
end
```

γ

δ

(Static) *Design Time*:

- *original_pre* ⇒ *new_pre* should be proved as a tautology
- *new_post* ⇒ *original_post* should be proved as a tautology

(Dynamic) *Runtime*:

- *original_pre* ∨ *new_pre* is checked
- *original_post* ∧ *new_post* is checked

① org_pre ⇏ new_pre

↳ as long as org_pre is satisfied, no precondition violation

② new_post ⇏ org_post

↳ as long as org_post is violated, there's postcond. violation.

Even if these two logical implications are not implemented during compile time due to poor design, the compiler won't complain and it does not lead to a run time violation but it may only give shock to the user?

not accurate.

In practice

PARENT

r

require α

ensure β

CHILD

redefine r end

① r ≡ ② r

do

end

(not equivalent)

③

r

require else True.

Is this a good design?

α ⇒ True

≡ True.

RT: α ∨ True

≡ True.

↳ no input restriction at all

require else False

do

ensure then True

end

RT: β ∨ False

≡ β

RT: β ∧ True

≡ β.

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
```

children at top level

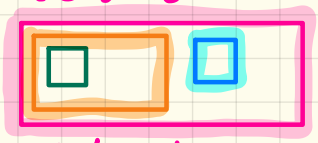
```
class WINDOW
  inherit
    RECTANGLE
    TREE[WINDOW]
end
```

```
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
```

You may:

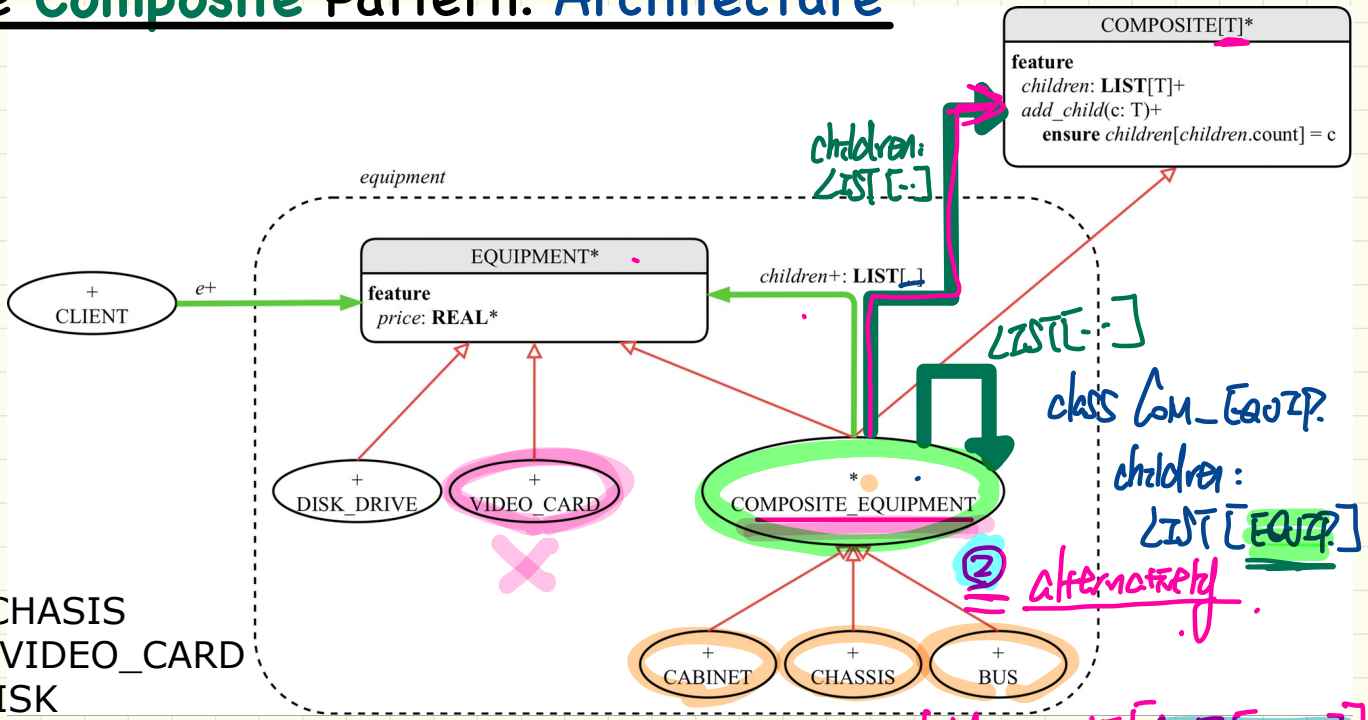
① rename 'descendants' to 'children'

② implement descendants to include all nested window



Why does w1.descendants have 2 items instead of 4 (w1, w2, w3, w4)?

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) ✗

Is `children: LIST[COMPOSITE]` a valid design?

Quiz8: Polymorphic Return Values

```
class
    B
inherit
    G
        redefine fa end
feature
    fa: STRING
        do
            Result := "B.fa"
        end
    fb: STRING
        do
            Result := "B.fb"
        end
end
```

```
class
    D
inherit
    A
        redefine fa end
feature
    fa: STRING
        do
            Result := "D.fa"
        end
    fd: STRING
        do
            Result := "D.fd"
        end
end
```

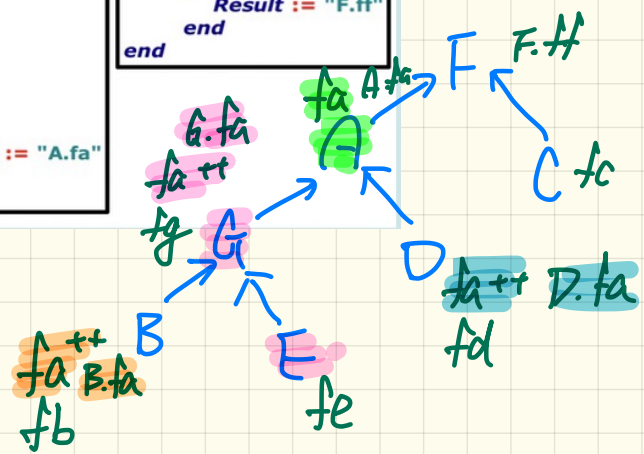
```
class
    G
inherit
    A
        redefine fa end
feature
    fa: STRING
        do
            Result := "G.fa"
        end
    fg: STRING
        do
            Result := "G.fg"
        end
end
```

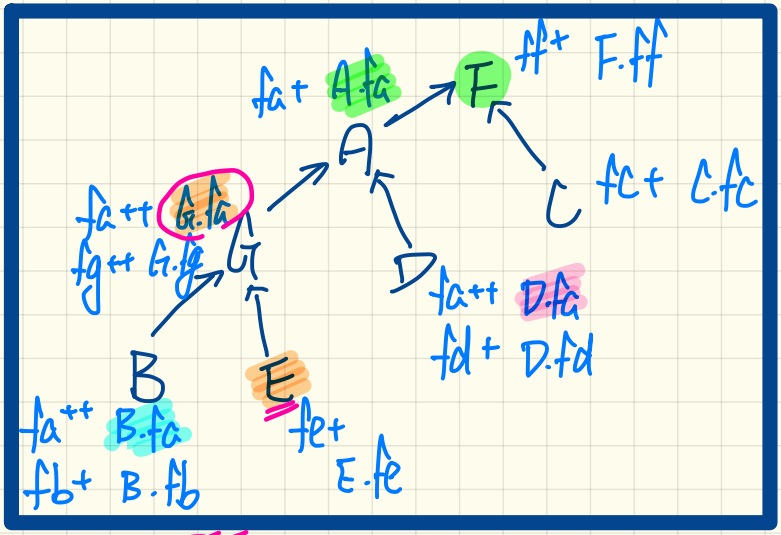
```
class
    C
inherit
    F
feature
    fc: STRING
        do
            Result := "C.fc"
        end
end
```

```
class
    E
inherit
    G
feature
    fe: STRING
        do
            Result := "E.fe"
        end
end
```

```
class
    A
inherit
    F
feature
    fa: STRING
        do
            Result := "A.fa"
        end
end
```

```
class
    F
feature
    ff: STRING
        do
            Result := "F.ff"
        end
end
```





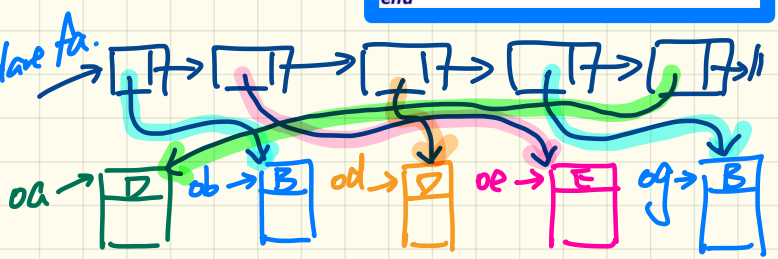

```
class
    CLIENT_2
create
    make
feature -- Collection
    a: LIST[A]
    make
        do
            create {LINKED_LIST[A]} a.make
        end
feature -- Routines
    add_a (obj: A)
        do
            a.extend (obj)
        end
    get_a (i: INTEGER): A
        require
            1 <= i and i <= a.count
        do
            Result := a[i]
        end
    get_f (i: INTEGER): F
        require
            1 <= i and i <= a.count
        do
            Result := a[i]
        end
end
```

```
oa: A
ob: B
od: D
oe: E
of: F
og: G
client: CLIENT_2

create {D} oa.make
create {B} ob.make
create {D} od.make
create {E} oe.make
create {B} og.make
create client.make

client.add_a (ob)
client.add_a (oe)
client.add_a (od)
client.add_a (og)
client.add_a (oa)
```

client.get_a(2).fa	Returns "G.fa"
client.get_f(2).fa	Invalid
client.get_a(5).fa	Returns "D.fa"
client.get_a(4).fa	Returns "B.fa"
client.get_a(1).fa	Returns "B.fa"
od := client.get_a(5)	Invalid
client.get_a(3).fa	Returns "D.fa"
og := client.get_a(1)	Invalid
client.get_f(4).fa	Invalid
of := client.get_a(4)	Valid



polymorphic collection