Lecture 10

Part 1

Contracts - Require Less vs. Ensure More

Assertions: Weak vs. Strong

$x > 3$

$x > 4$

Assertions: Preconditions

```
withdraw_v1(amount: INTEGER)
    require
      P1: amount > 0
```

```
withdraw_v2(amount: INTEGER)
    require
      P2: amount ≥ 0
```

Example Input
100
0
-100

Assertions: Postconditions

Example Input

79

-34

62

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

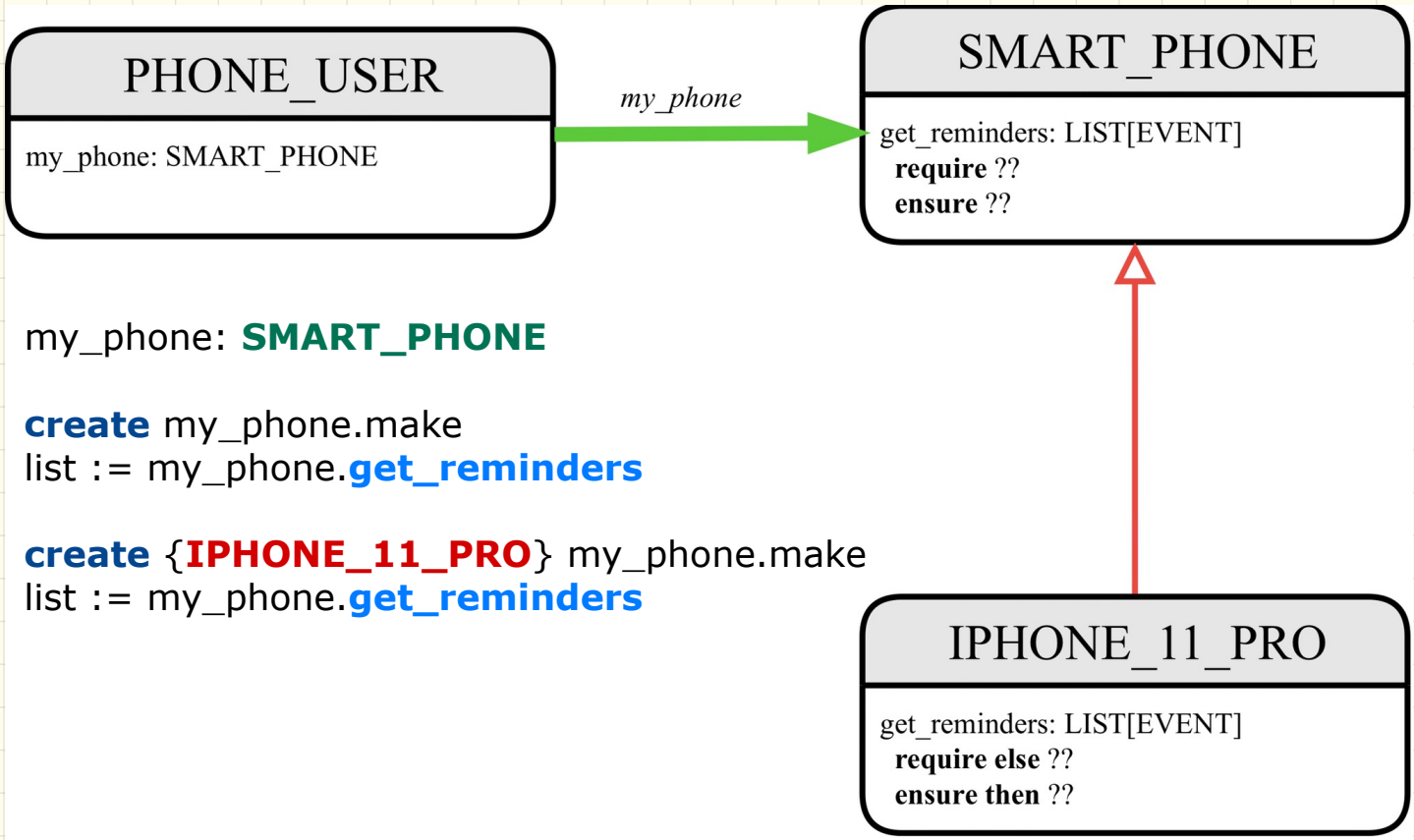
```
f2(i: INTEGER): BOOLEAN
    ensure
      Q2: Result = (i > 0) ∧ (i mod 2 = 0)
```

Lecture 10

Part 2

Inheritance & Contracts - Static Analysis

Subcontracting: Architectural View



my_phone: **SMART_PHONE**

create my_phone.make
list := my_phone.**get_reminders**

create {**IPHONE_11_PRO**} my_phone.make
list := my_phone.**get_reminders**

Subcontracting: Example (1)

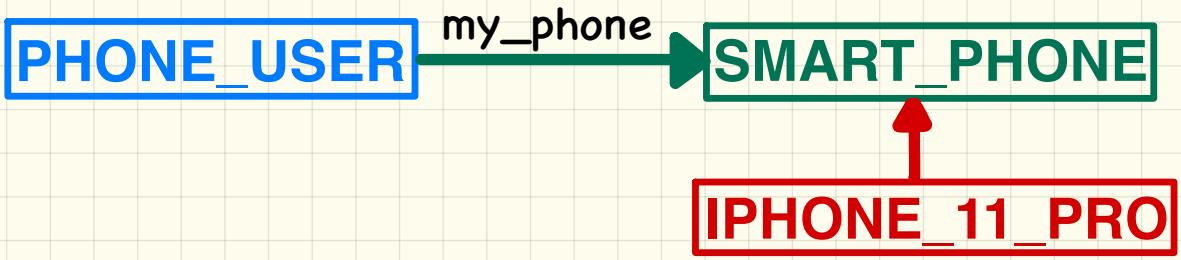
```
class SMART_PHONE
  get_reminders: LIST[EVENT]
    require
      α: battery_level ≥ 0.1 -- 10%
    ensure
      β: ∀e : Result | e happens today
end
```

```
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
```

```
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)

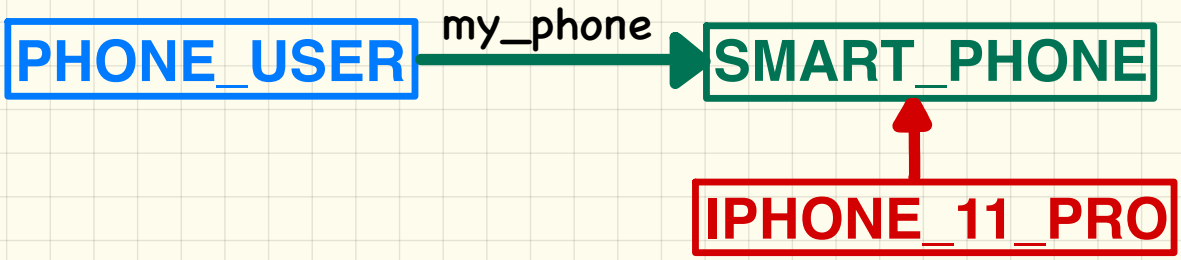
```
class SMART_PHONE
  get_reminders: LIST[EVENT]
    require
      α: battery_level ≥ 0.1 -- 10%
    ensure
      β: ∀e : Result | e happens today
end
```

```
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
```

```
my_phone: SMART_PHONE

create my_phone.make
list := my_phone.get_reminders

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



Lecture 10

Part 3

Inheritance & Contracts - Runtime Checks

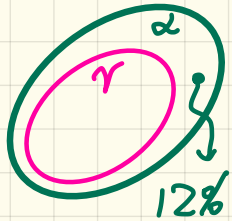
Subcontracting: Example (1)

```
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

$\because \gamma \Rightarrow \alpha$

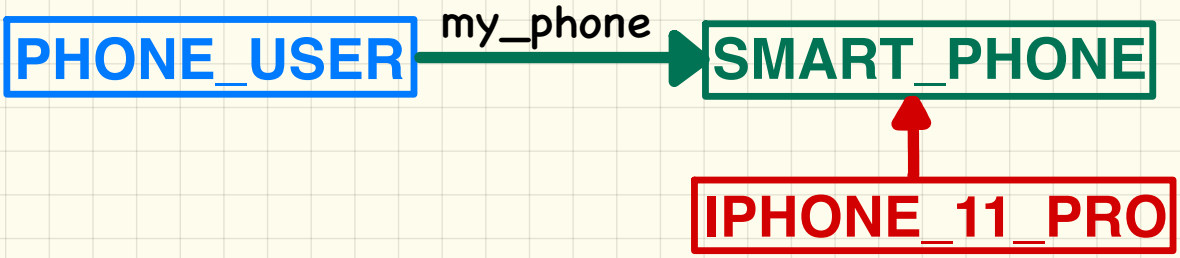


Runtime :

```
my_phone: SMART_PHONE

create my_phone.make
list := my_phone.get_reminders

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



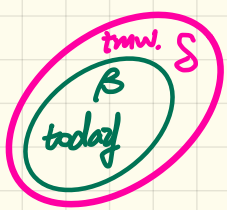
Subcontracting: Example (2)

```
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
```

Invalid

$\because \beta \Rightarrow \delta$

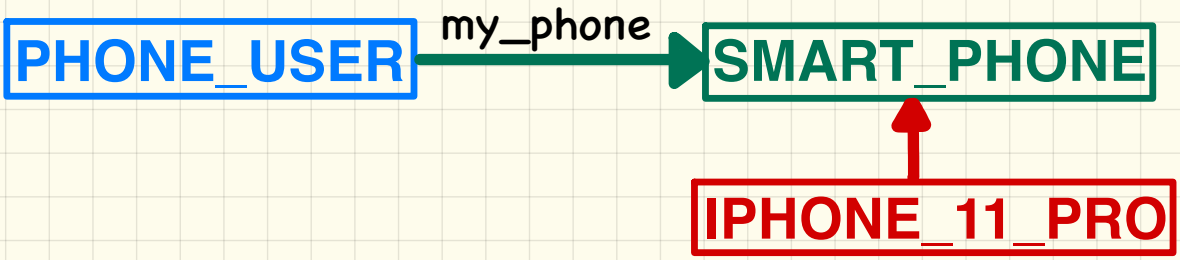


Runtime:

```
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
```

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
```

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
```

Contract Re-Declaration:

Missing Post-Condition in Descendant

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

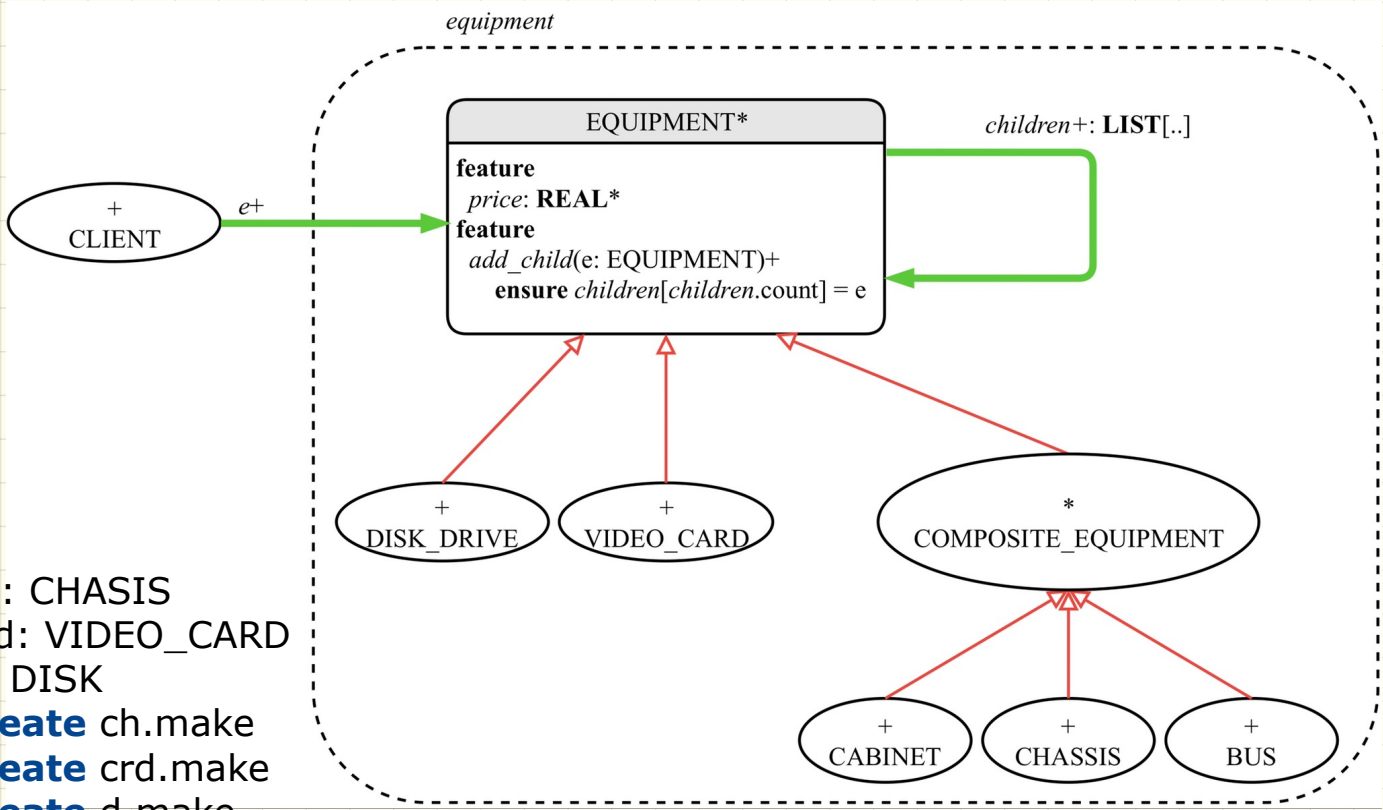
```
class BAR
inherit FOO redefine f end
  f
    do ...
    end
end
```

Lecture 10

Part 4

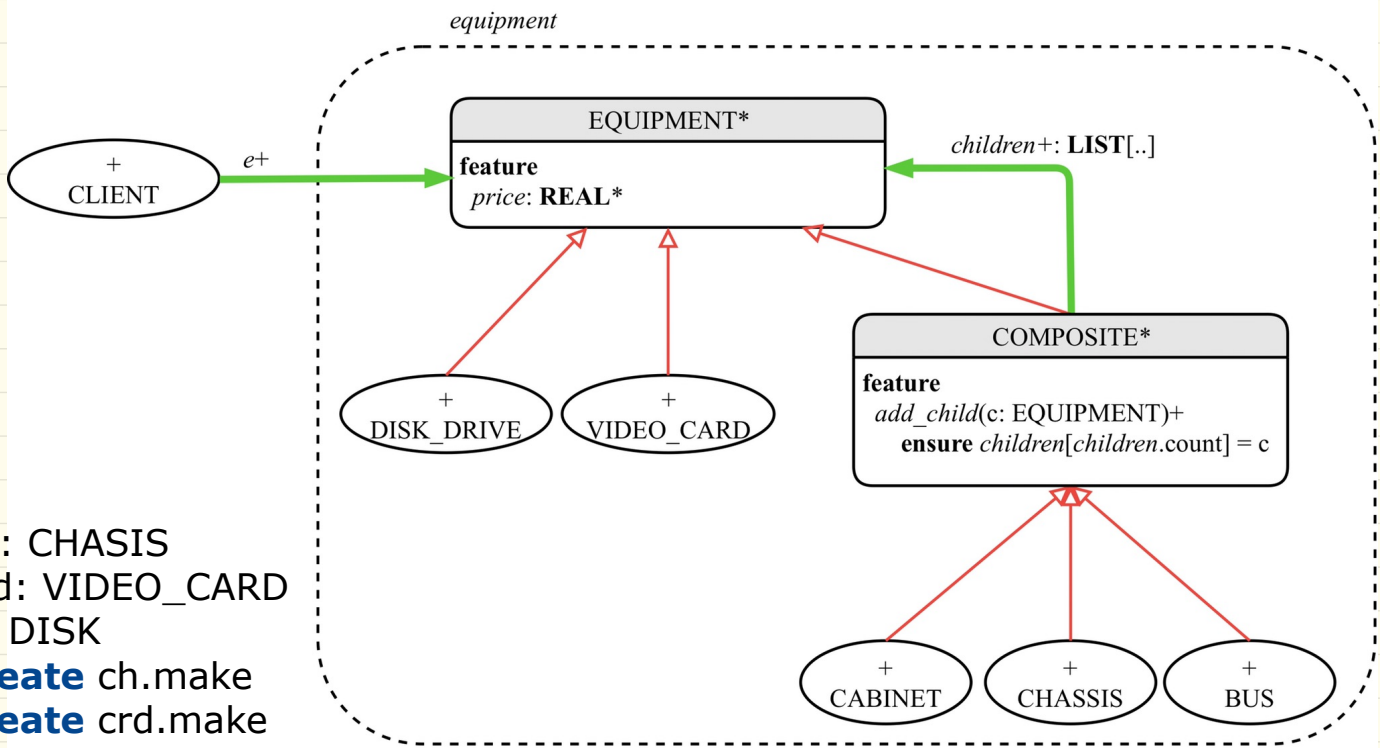
Recursive Systems - Design Attempts

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



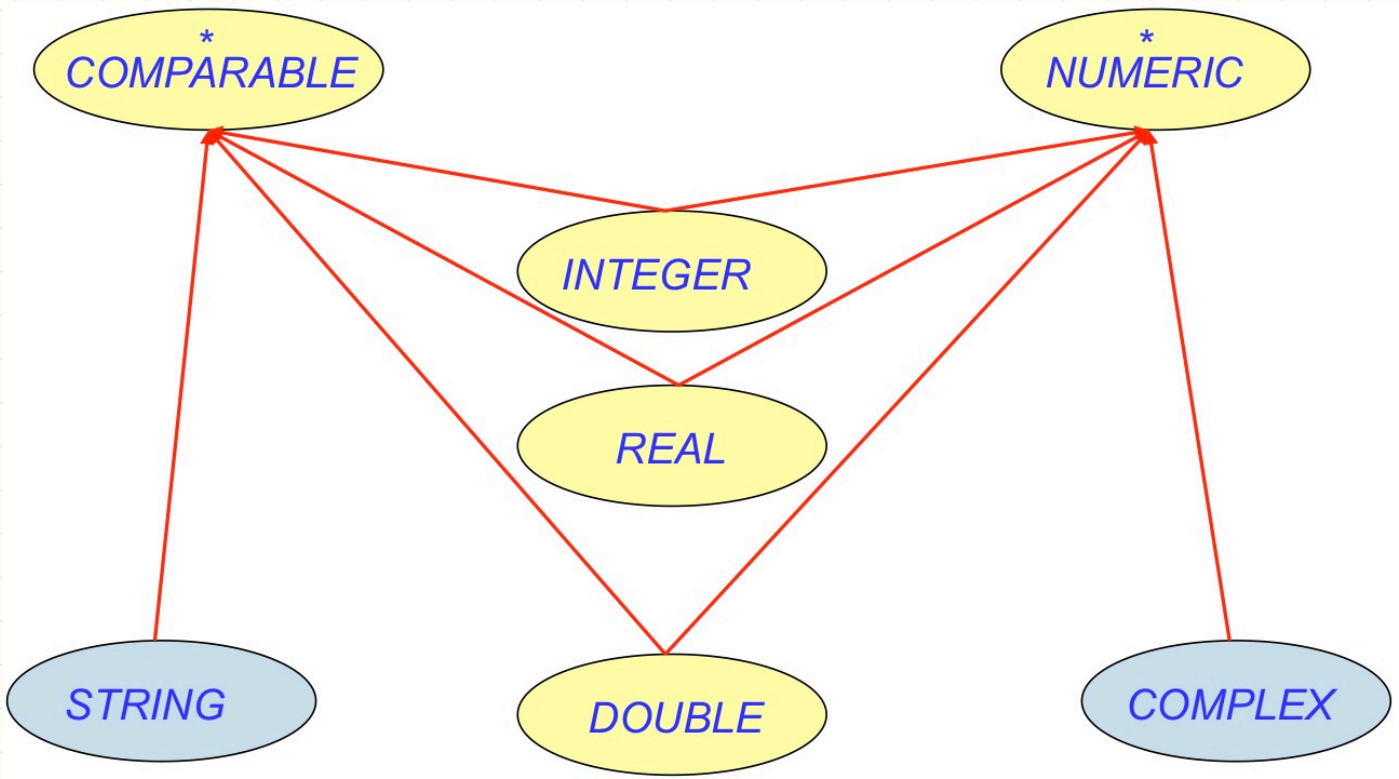
ch: CHASIS
crd: VIDEO_CARD
d: DISK
create ch.make
create crd.make
create d.make
ch.add_child(crd)
ch.add_child(d)
crd.add_child(d)

Lecture 10

Part 5

Multiple Inheritance

Multiple Inheritance: Example



Multiple Inheritance: Exercise

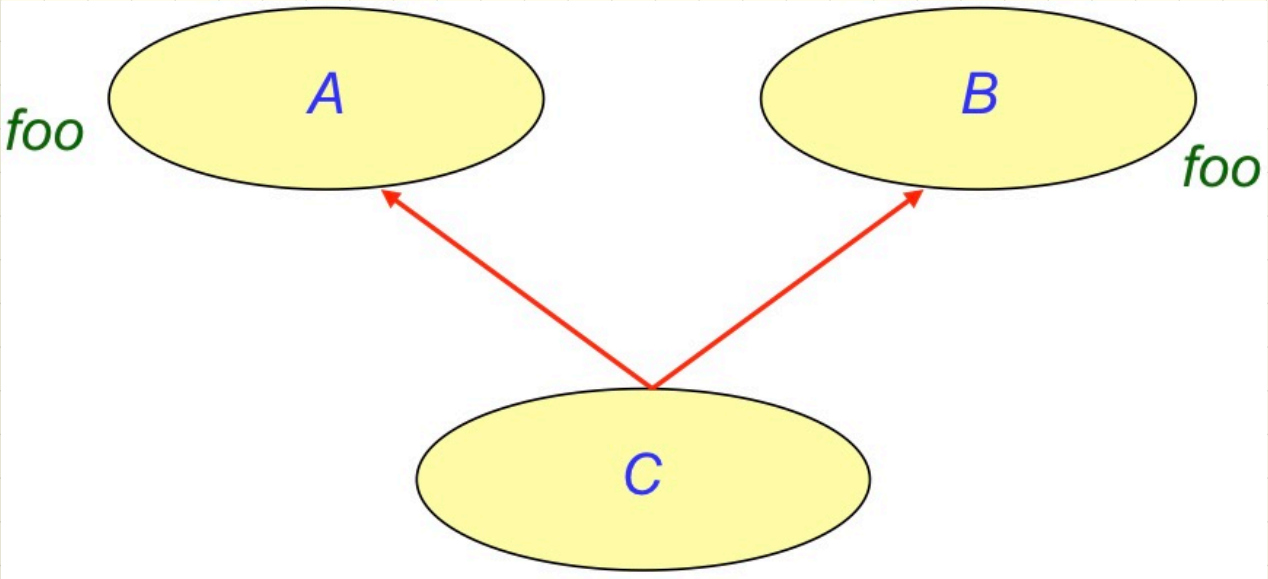
```
class RECTANGLE
  feature -- Queries
    width, height: REAL
    xpos, ypos: REAL
  feature -- Commands
    make (w, h: REAL)
    change_width
    change_height
    move
end
```

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

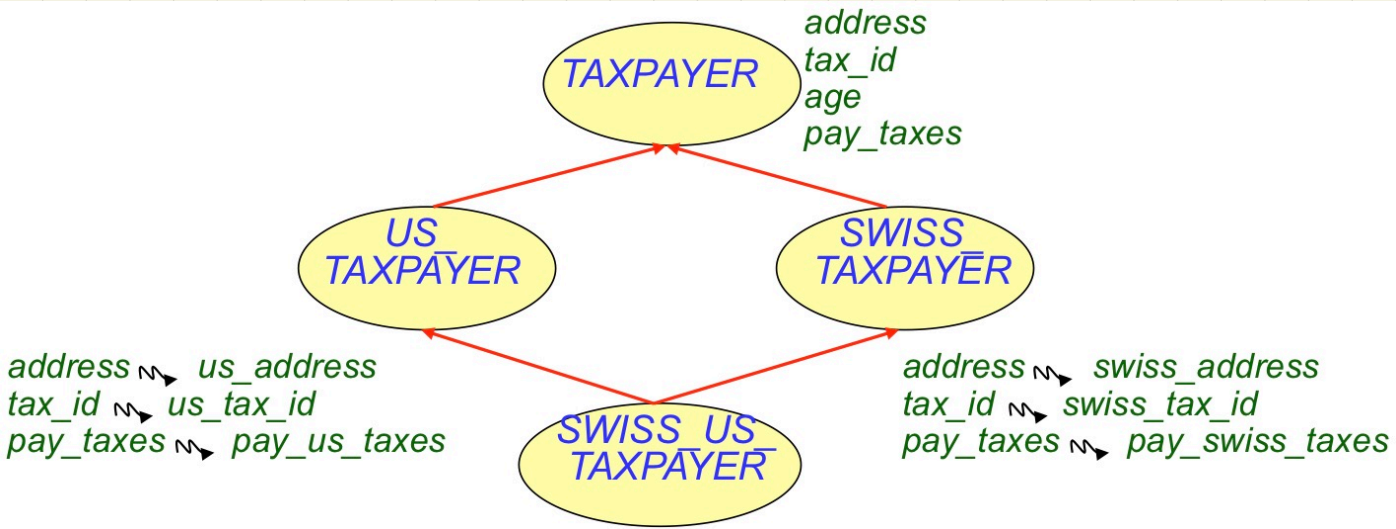
```
class WINDOW
  inherit
    RECTANGLE
    TREE[WINDOW]
end
```

```
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



Multiple Inheritance: Name Clashes

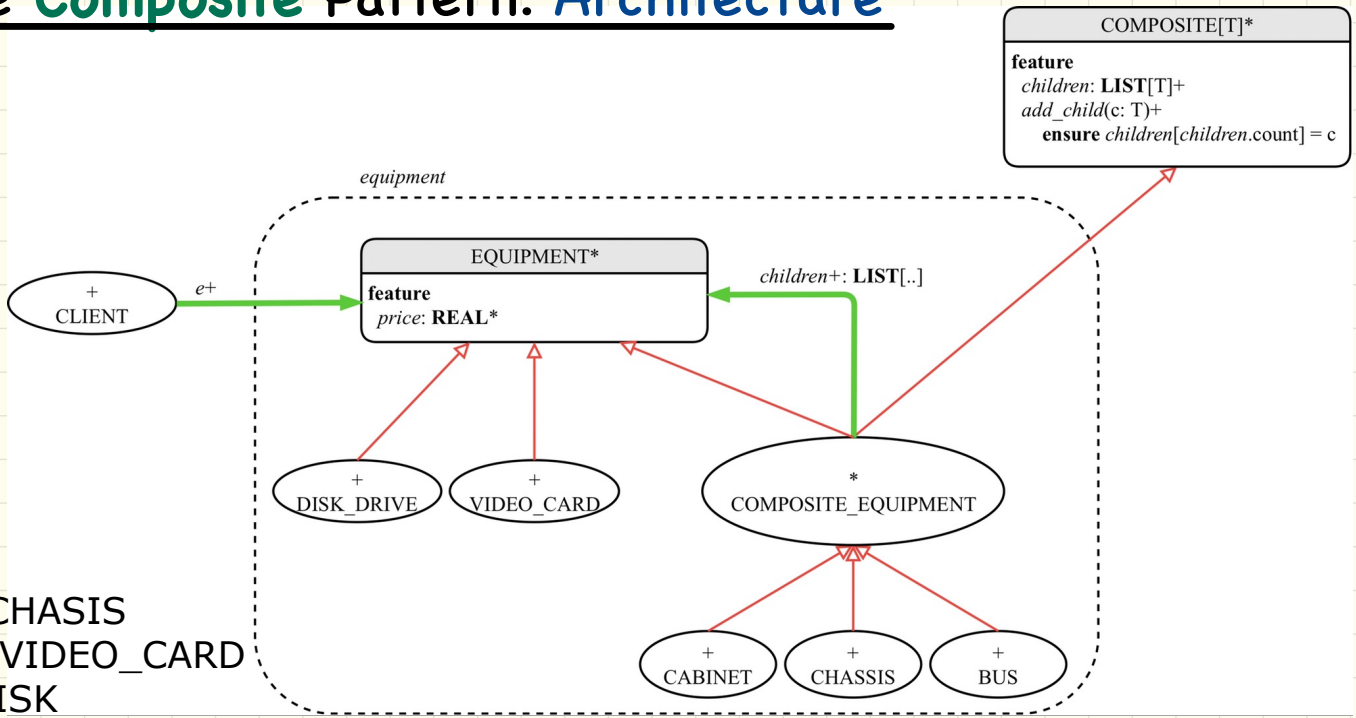


Lecture 10

Part 6

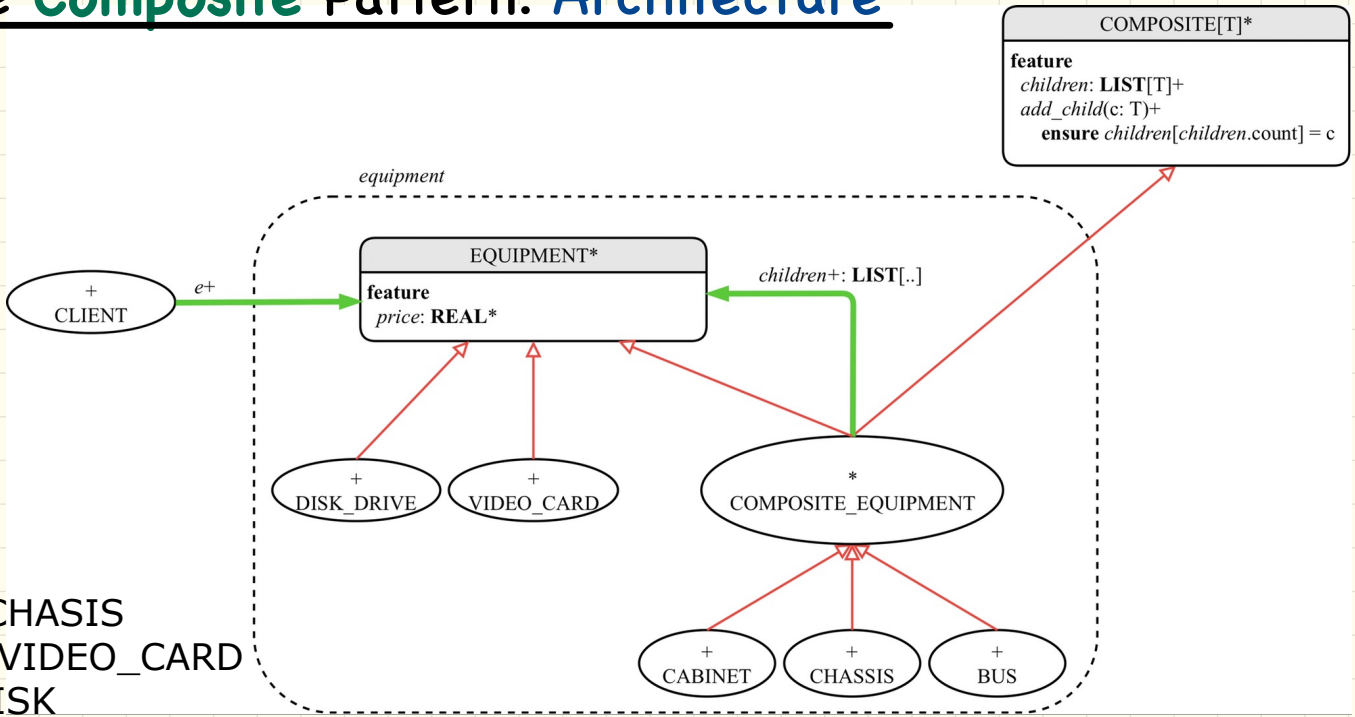
Composite Design Pattern

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



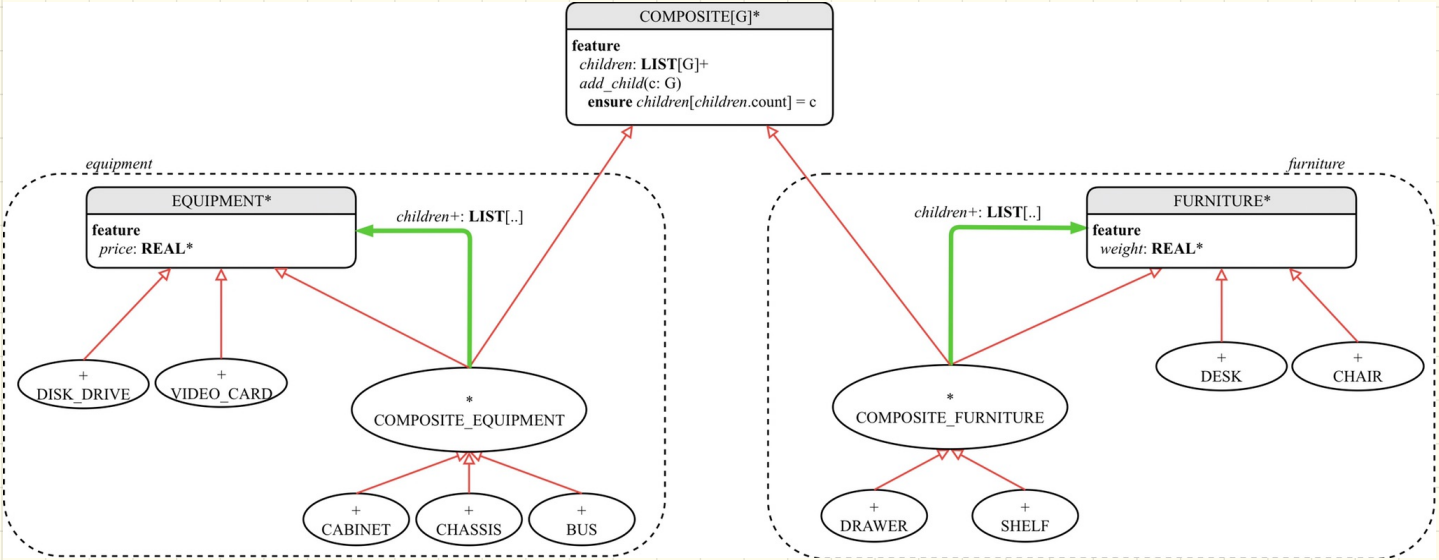
ch: CHASIS
crd: VIDEO_CARD
d: DISK

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?

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
```

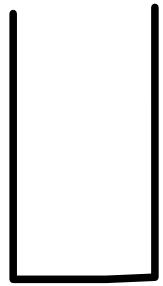
```
class
  CARD
inherit
  EQUIPMENT
feature {NONE}
  unit_price: REAL
feature
  make (n: STRING; p: REAL)
    do name := n ; unit_price := p end
  price
    do Result := unit_price end
end
```

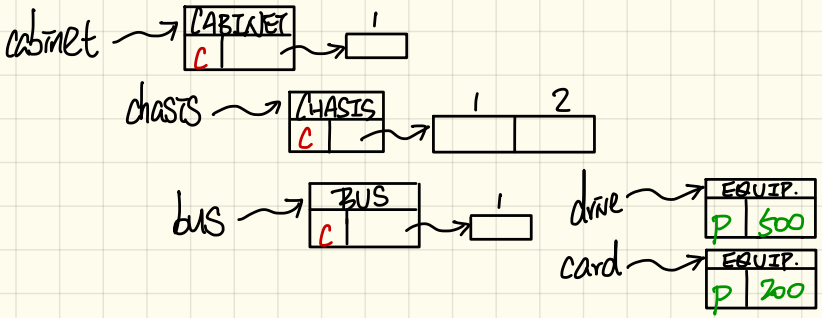
```
class
  COMPOSITE_EQUIPMENT
inherit
  EQUIPMENT
  COMPOSITE [EQUIPMENT]
create
  make
feature
  make (n: STRING)
    do name := n ; create children.make end
  price : REAL -- price is a query
      -- Sum the net prices of all sub-equipments
    do
      across
        children is c
      loop
 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