The Composite Design Pattern

EECS3311: Software Design
Fall 2017

CHEN-WEI WANG

Motivating Problem (1)

- Many manufactured systems, such as computer systems or stereo systems, are composed of ***individual components*** and ***sub-systems*** that contain components.
  e.g., A computer system is composed of:
  - Individual pieces of equipment (*hard drives*, *cd-rom drives*)
    Each equipment has *properties*: e.g., power consumption and cost.
  - Composites such as *cabinets*, *busses*, and *chassis*
    Each *cabinet* contains various types of *chassis*, each of which in turn containing components (*hard-drive*, *power-supply*) and *busses* that contain *cards*.
- Design a system that will allow us to easily *build* systems and *calculate* their total cost and power consumption.

Motivating Problem (2)

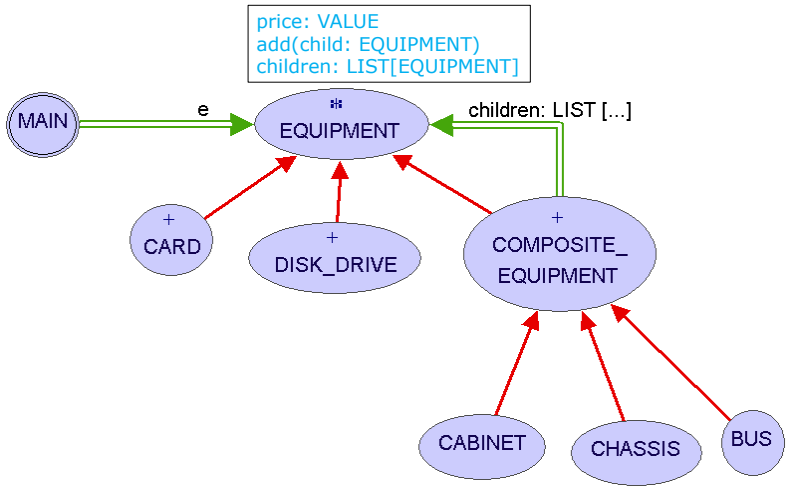
Design for *tree structures* with whole-part *hierarchies*.

Challenge: There are ***base*** and ***recursive*** modelling artifacts.

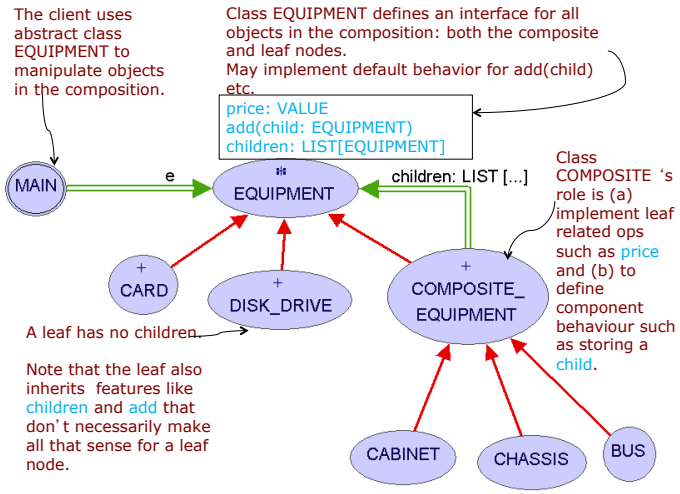
Solution: The Composite Pattern

- **Design**: Categorize into *base* artifacts or *recursive* artifacts.
- **Programming**:
  Build a *tree structure* representing the whole-part *hierarchy*.
- **Runtime**:
  Allow clients to treat *base* objects (leafs) and *recursive* compositions (nodes) *uniformly*.
  ⇒ *Polymorphism*: *leafs* and *nodes* are "substitutable".
  ⇒ *Dynamic Binding*: Different versions of the same operation is applied on *individual objects* and *composites*.
  e.g., Given e: ***EQUIPMENT***:
  - `e.price` may return the unit price of a ***`DISK_DRIVE`***.
  - `e.price` may sum prices of a ***`CHASIS`***' containing equipments.

Composite Architecture: Design (1.1)



Composite Architecture: Design (1.2)



Composite Architecture: Design (1.3)

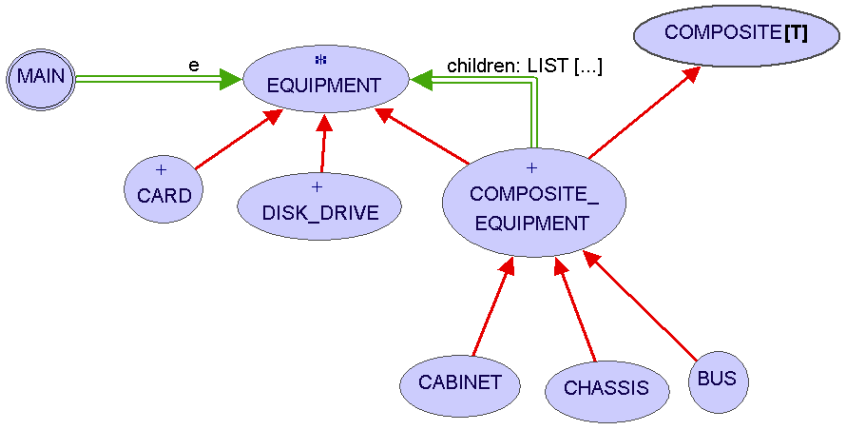
Q: Any flaw of this first design?

A:

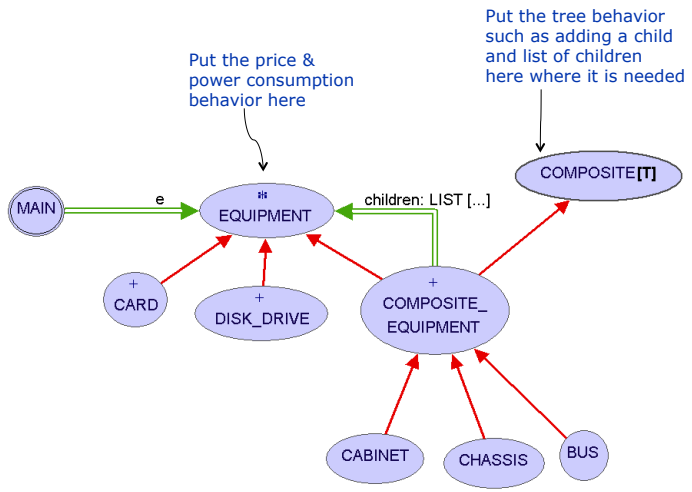
The `add(child: EQUIPMENT)` and `children: LIST[EQUIPMENT]` features are defined at the `EQUIPMENT` level.

⇒ Inherited to all *base* equipments (e.g., `HARD_DRIVE`) that do not apply to such features.

Composite Architecture: Design (2.1)



Composite Architecture: Design (2.2)



Implementing the Composite Pattern (1)

```
deferred class
  EQUIPMENT
feature
  name: STRING
  price: REAL -- uniform access principle
end
```

```
class
  CARD
inherit
  EQUIPMENT
feature
  make (n: STRING; p: REAL)
    do
      name := n
      price := p -- price is an attribute
    end
end
```


Implementing the Composite Pattern (2.1)

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

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

Exercise: Make the `COMPOSITE` class *iterable*.

Implementing the Composite Pattern (2.2)

```
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 as cursor
      loop
        Result := Result + cursor.item.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
```


Index (1)