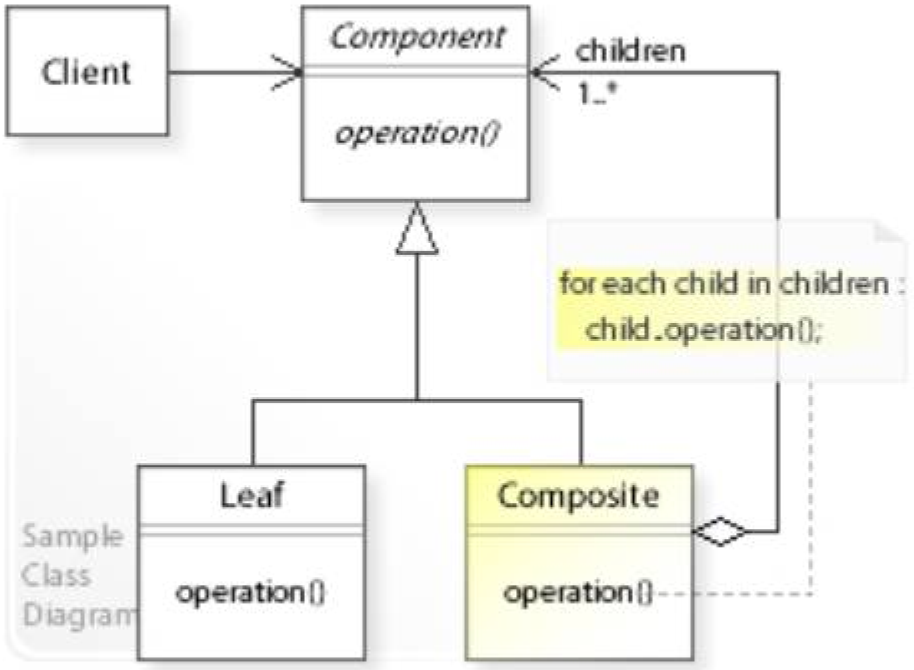
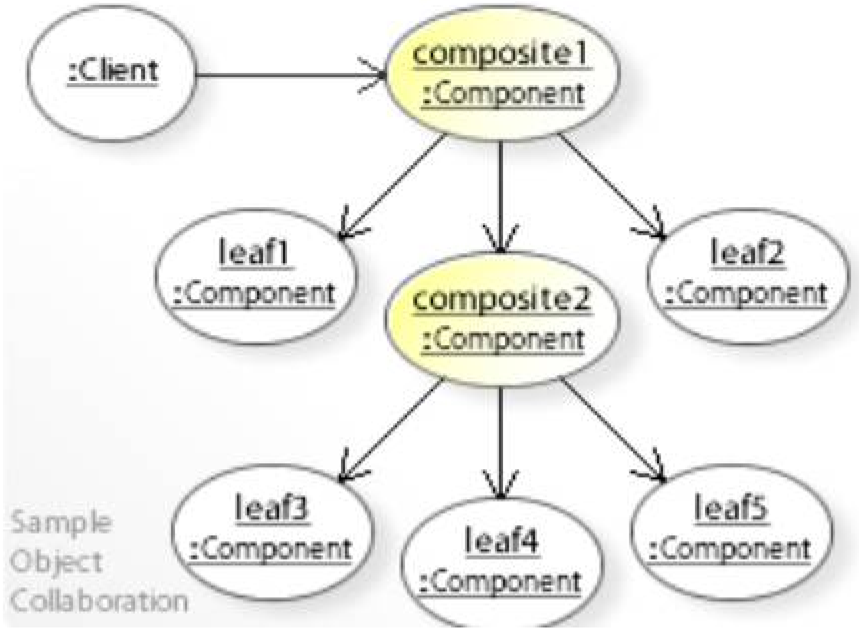
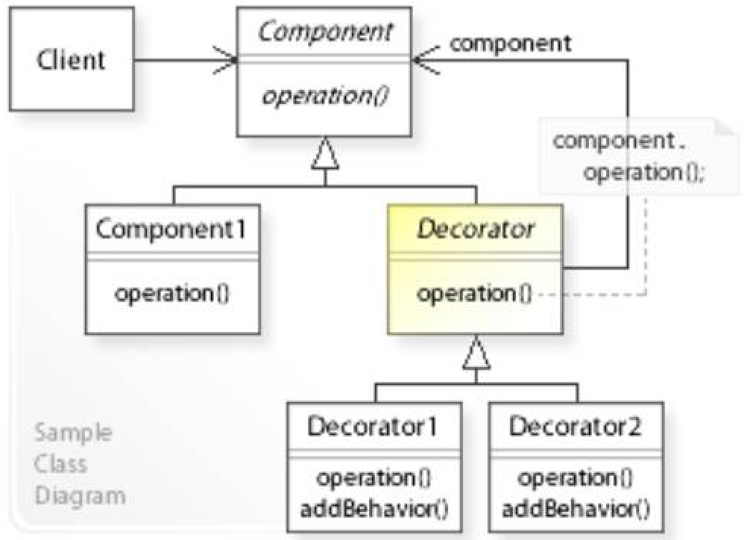
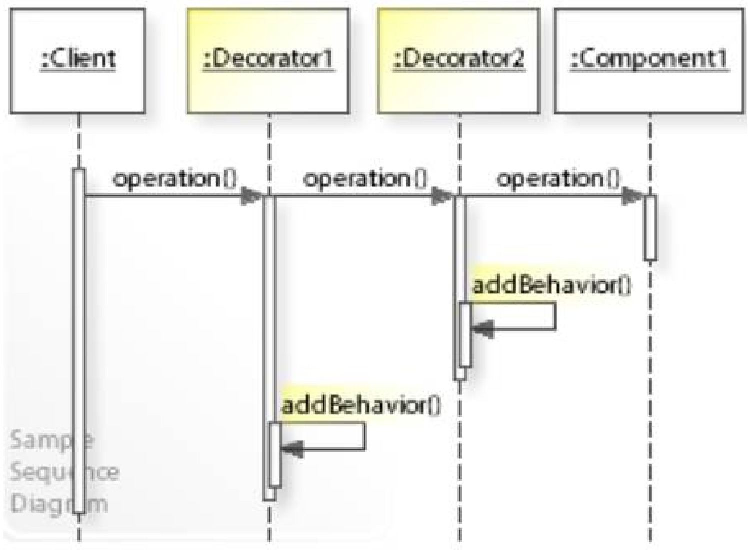
Design Patterns – Recursive Structures
Composite (**AKA Tree**)
 Pro: unifies Leaf & Non-leaf classes
 o- Client uses IComponent = INode
 o- Leaf node & Composite Non-leaf node inherit INode
 o- Non-Leaf node has kid_refs to INode
 o- Non-Leaf node has add/del ops, too.
 o- Canon Ex: Graphics display of drawn shapes, nested.





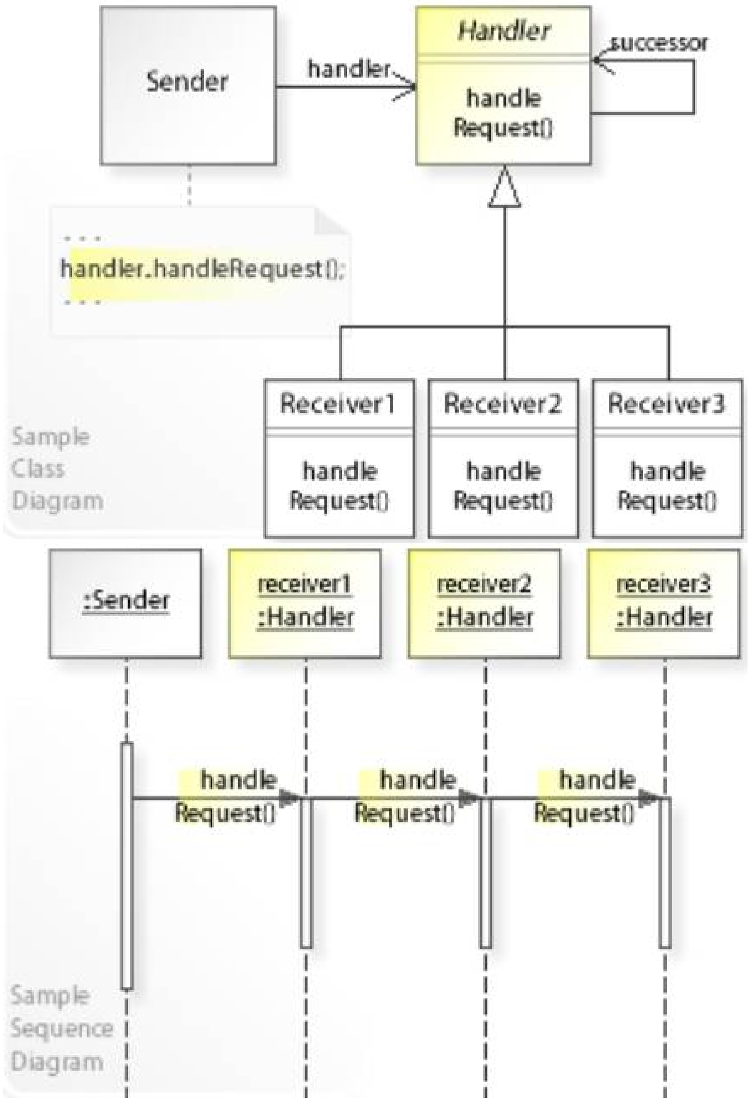
Decorator (List – "Decorate" Target
with extra behavior before and/or after)
- o- Works exactly like Composite, but for diff purpose
- o- Client uses IComponent = INode
- o- Leaf node & Composite/Decorate Non-leaf node inherit INode
- o- Non-Leaf node itself has INon-Leaf cutout cuz diff kinds
- o- Non-Leaf node has kid_ref (1) to INode
- o- Non-Leaf node has add/del ops, too.
- o- Leaf node is Target object w/ some operation to decorate
- o- Non-Leaf nodes see operation call before & after Leaf
- o- Canon Ex: Window display + extras: scroll bars, title bar, borders, …



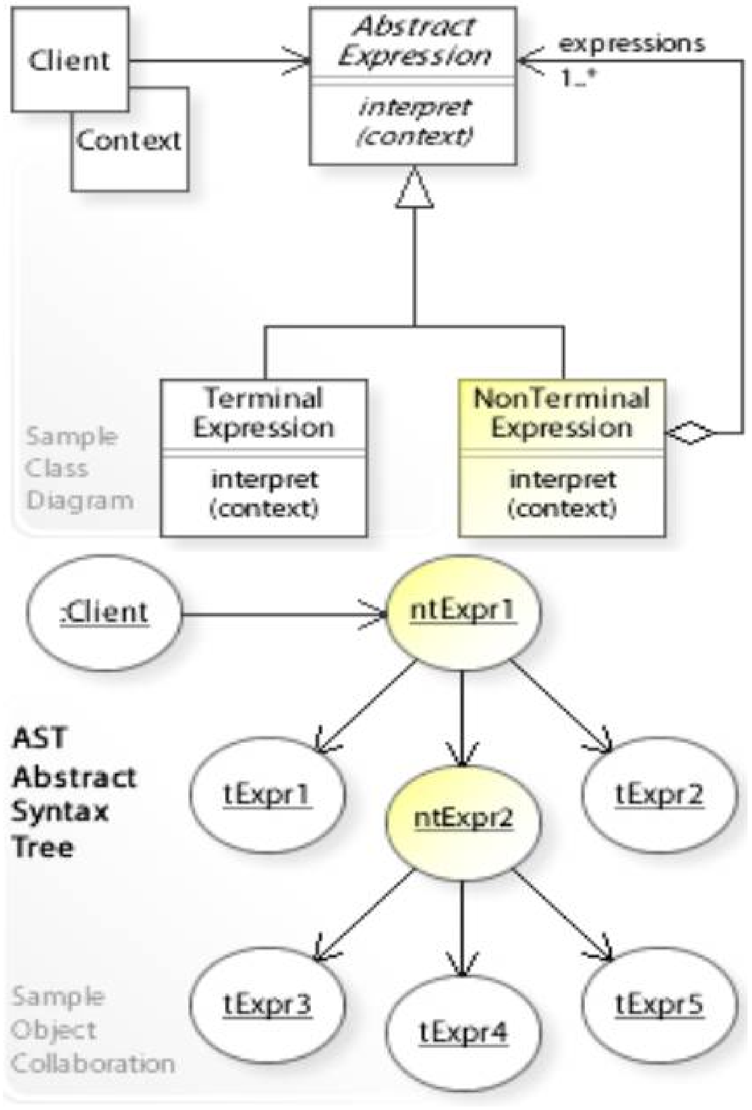


CECS 434 SWE – Dsgn Pats.Recur+   Wk #13 vLect #1 4/15 p 1

Chain of responsibility (List – No final "Target",
 plus only 1 ob does the operation)
- o- Works exactly like Composite, but for diff purpose
- o- Client uses IHandler = INode
- o- Leaf node & Composite Non-leaf node inherit Inode
  - o-- Here, they didn't cutout Handler ("shame on them")
- o- Non-Leaf node itself has INon-Leaf cutout cuz diff kinds
- o- Non-Leaf node has kid_ref (1) to INode
- o- Non-Leaf node has add/del ops, too.
- o- No need for Leaf node for Target object w/ some operation to decorate
- o- Non-Leaf nodes see operation call & decides should handle it?
- o- Canon Ex: Exception handling for some Throw



Interpreter (**AKA AST/Abstract Syntax Tree -**
implements a specialized language (AKA **DSL**))
- o- Works exactly like Composite, but for diff purpose
- o- Client uses IAbst_Expr = INode
- o- Leaf node & Terminal_Expr Non-leaf node inherit Inode
  - o-- Here, they didn't cutout Non-leaf (they should have)
- o- Non-Leaf node itself has INon-Leaf cutout cuz diff kinds
- o- Non-Leaf node has kid_ref (1) to INode
- o- Non-Leaf node has add/del ops, too.
- o- Nodes each control how operation works for their kids
- o- Canon Ex: DSL (AKA Domain-Specific Language)



CECS 434 SWE – Dsgn Pats.Recur+   Wk #13 vLect #1 4/15 p 1

o13.3.1 Arch Styles p 258

Data-Centered Arch
 Big DB, many **CRUD** clients
 Issues:
  ACID: Atomicity Consistency Isolation Durability
   Atomicity: all or nothing rule (parts of xtn)
   Consistency: only valid data in db (wrt constraints)
   Isolation: seq of overlapping xtns can't interfere w each other
   Durability: DB atomicity if crash during xtn

  BASE: (Wordplay on "ACID" from Chemistry) [**Nox**]
   Basic Availability (AKA local availability)
   Soft-state (AKA inconsistent for newest changes)
   Eventual consistency (AKA consistent for older changes)
   o- Badly chosen expressions (hard to remember)

  CAP theorem:
   Consistency: all processors see same thing
     no Avail: (by delaying local access till remote change is here too)
     no Partition: (by having no remote processors)
   Availability: distant changes available immediately
     (by delaying local access)
   can't have **Consistency**, **Availability**, and **Partition** tolerance,
   you have to settle for two out of three.
   (Eric Brewer) [**Nox**]