

## GoF Patterns (continued)

- Today: Continue chapter 23 (GoF patterns)
- The Composite Pattern is covered in detail
  - ...what's it about? (section 23.7)
  - (See box, page 359)
- Summary:
  - We want to be able to treat the same way
    - a bunch of similar objects, and
    - a single object

## GoF Composite Pattern

- General idea:
  - define a "bag" object
  - It contains a bunch of individual objects
  - It has an API (Application Programmer Interface)
- Its API is the same as that of other objects
  - (these may be "non-bag" objects)
- Let's start saying "composite object"
  - (instead of "bag object")

## Composite and Other Patterns

- The Adapter Pattern also involved clumping things
- The Strategy Pattern also involved clumping things
- The Composite pattern also involves clumping things
- Clearly, clumping is a powerful concept...
- We'll look at some details...
  - ...then compare the composite with the strategy & adapter patterns
- The key new idea of the Composite Pattern:
  - Same interface to
    - Composite (bag) object(s), and
    - Single object(s)
- An example that connects well to previous discussions:
  - Pricing strategies for a POS system
  - Some interesting differences emerge compared to other patterns...

## An Example for the Composite Pattern

- Recall the POS system and its pricing policies
- What if there are multiple pricing policies that can conflict?
  - What do you do then? And, how should software do it?
- Example pricing policies that could conflict
  - 20% senior discount policy
  - 20.1% student discount policy
    - What if someone is a senior and a student?
  - Member discount of 5%
  - Monday special: \$5 off purchases exceeding \$50
    - Someone buying \$51 of stuff on Monday could be a senior, a student, & a member
- Let's list some ambiguities and/or conflicts that might arise in calculating the total for a sale...

## Resolving the Conflicts

- First, there is a policy issue to be determined
  - Use the policy giving the biggest discount
  - Use the policy the customer requests
  - Apply each policy in sequence
    - Could the order of the sequence make a difference?
- Then, there is a software design issue to be determined

## Resolving the Conflicts II

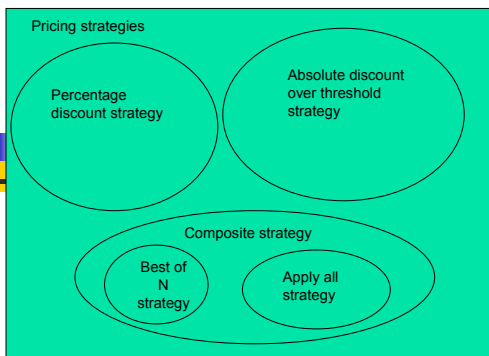
- First, there is a policy issue to be determined
- Then, there is a software design issue to be determined
  - We'd like for the object of class Sale to just
    - Request a discount policy object
    - Apply it without worrying about what's in it
      - (could be one policy, a set of policies, etc.)
  - For Sale to have to deal with those details would be
    - High in coupling
    - Low in cohesion

## Resolving the Conflicts III

- We'd like for the object of class Sale to just
  - Request a discount policy object
  - Apply it without worrying about what's in it
    - (one policy, a set of policies, etc.)
- The Composite Pattern says...
  - that is a good thing to do, and also says...
  - ...how to do it in an object-oriented way

## Resolving the Conflicts IV

- The Composite Pattern says...
  - that is a good thing to do, and
  - how to do it in an object-oriented way
- Here's how...
  - Define a base class for composite pricing strategies
  - It has subclasses, each some composite strategy
  - It conceptually keeps the software design clear



## Resolving the Conflicts V

- Composite, absolute, and percentage are now each a policy
- Given the composite strategy
  - ...there are more decisions to make next
- Conceptually, which is a better way to classify colors?
  - 1) Colors=
    - Yellow, purple, greenish blue, purplish blue, sky blue, navy blue
  - 2) Colors=
    - Yellow, purple, blue
      - Blues=greenish blue, purplish blue, sky blue, navy blue

## A Design Using the Composite Pattern

- Let's look at Figure 23.13
  - Also look at Java code, p. 362
- Recall the colors analogy
- Note that, to Sale,
  - a call to an atomic pricing strategy
  - is done the same way as
  - a call to a composite strategy
    - ...even though...
    - the composite strategy uses >1 atomic strategies
- Note the subclass UML notation

## A Design Using the Composite Pattern II

- Let's look at collaboration, Figure 23.14
- Note the :Object boxes
  - UML way to not specify the class
  - But the interface is specified
  - Java note – there actually is an Object class...
- Any disadvantage to the design of the Figure?
- See Figure 23.15, note UML notation



## Composite Vs. Strategy, and Adapter Patterns

- Strategy vs. composite:
  - Compare Figures 23.8 with 23.13...
- Adapter vs. composite:
  - Adapter: use a stable interface to mediate
    - between
      - diverse and/or changing interfaces to related modules  
...and...
      - the rest of the system
  - Composite pattern hides conceptual variations
  - Adapter pattern hides interface variations