



## GoF Strategy Pattern in More Detail

- Reference: text starting at section 23.6



## POS Example: Why Care?

- 10/29/03 (AP) story:
- IBM has developed a "Smart Shopping Cart" prototype
- Dan Hopping, IBM: "We'll see more change in the next five years in the way people shop than in the last 20."
- It's a traditional shopping cart, with more stuff
- It has a computer device and card reader mounted on it, bar code reader
- If you swipe your card, it can do things like
  - Remind you to buy a necessity you haven't gotten in a while
  - Suggest things to buy based on past purchases
  - Remind you to make sure you've used up specific perishables bought on the last visit (and suggest you buy more)
- This is just a start



## POS Example: Why Care? II

- The cart could also have a bar code scanner...
  - (or some other technology)...
- so the cart could know what's in it
- If you pick up peanut butter, it will suggest jelly
- It will tell you where the jelly is...



## POS Example: Why Care? III

- The cart could ...
  - Let you key in the word "pizza"
    - And tell you where it is
  - Read radio frequency identification chips instead of bar codes (why is that good?)
  - Not only read/write your card, but
    - ...upload and archive its contents to the Dept. of Homeland Security for "security purposes"
      - (or maybe just to the store headquarters for their purposes)



## POS Example: Why Care? IV

- Since the cart could know what is in it...
  - It could decrement your account, accept cash in a slot, or charge your credit card on request
  - You could leave the store without going through a checkout line
  - The store could send you an email when something you looked for is delivered to the store
  - Anything else?



## On to the GoF Strategy Pattern...

## Strategy Pattern

- Example of a kind of problem:
  - Pricing can change frequently
  - A sale can last for 1 afternoon and involve 10% off everything
  - Sales tax on school items can be 0 for certain days
  - Etc., etc. – as mentioned, I'm not a big shopper
    - ...but feel free to use your imagination

## Strategy Pattern II

- Problem:
  - Design for *varying, related computations*
  - That is, make it easy to change the computation
- Solution:
  - Put each variation in its own class/module
  - Make a single interface to all of them

## Strategy Pattern III

- Solution:
  - Put each variation in its own class
  - Make a single interface to all of them
- Each variation is a "strategy"
- The interface is an example use of the *adapter* design pattern

## Strategy Pattern III

- Solution:
- Put each pricing strategy in its own class
  - Details like constants can be
    - ...read from a file
    - ...used to fill out an instance of a strategy class
    - ...so the actual pricing policy is in an object
      - (not a class)
  - Strategy – a general outline of action
  - Detailed policy – not part of the strategy itself

## Strategy Pattern IV

- Let's give each pricing strategy class...
  - ...a `getTotal(Sale s)` method
  - `getTotal()` gets info from `s` and applies a price modification algorithm
- Put each pricing strategy in its own class
  - Details like constants can be
    - ...read from a file
    - ...used to fill out an instance of a strategy class
    - ...so the actual pricing policy is in an object
      - (not a class)
  - Strategy – a general outline of action
  - Detailed policy – not part of the strategy itself

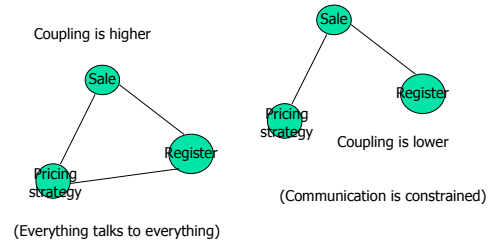
## Strategy Pattern V

- See Figure 23.8...
- UML notation issues:
  - Note the "stereotype" in "guillemets"
    - `<<` is not quite a guillemet
    - `«` (OOABx in unicode) is
    - `<<` `<<` etc. are close but not quite there
  - Note the dotted, empty-headed arrows
  - `ISalePricingStrategy` is not a UI, obviously
  - Notice the sample `getTotal()` codes
    - Where are the constants from? Maybe a file

## Strategy Pattern VI

- Sale probably sends a figure to Register
- Sale should probably manage pricing strategy calls (not Register)
- Why?
  - 1) The pricing strategy may need line item details which Sale has
  - 2) Information expert pattern says do the action where the information is
  - 3) So Sale not Register should invoke the strategy
  - 4) This promotes low coupling as well
    - (see figure, next slide)

## Information Expert Pattern Application Results in Low Coupling (Also a Design Pattern)



## Strategy Pattern VII

- (See Figure 23.9)
- Note the need for context object Sale to
  - pass itself (*this*, in Java)
  - ...to the pricing strategy object
- The strategy object *has parameter visibility to* the Sale object
- ...that means *can view as a parameter*
  - (Sale is the parameter to strategy)
  - (This is counter-intuitive use of English)
  - See Figure 23.10

## Strategy Pattern VIII

- Use of the singleton (GoF) pattern:
  - The pricing strategy factory should be a singleton
  - See Figures 23.11 and 23.12
  - Note in 23.12 that when
    - ..."a Sale instance is created, it can ask the factory for its pricing strategy"

## Strategy Pattern IX

- What about the details of the pricing policy?
  - ...the amount of percent to discount
  - ...the level at which a constant discount kicks in
  - ...the amount of the constant discount
  - ...the age at which senior citizen discount kicks in
- These are not part of the strategy, they are details
- They can be stored in a file
- The factory creates a strategy object that has specific details set (in its member variables) by the factory, which reads the file