



The Observer (Publish-Subscribe) Pattern

- Reference: section 23.9
- Problem: the UI should show updated data
- POS example: The sale total should be kept up-to-moment so customer can see how it is increasing



The Observer (Publish-Subscribe) Pattern II

- A "bad" solution:
 - Have Sale object send a message to the UI after adding in every SalesLineItem
 - Why bad?
 - Because of (p. 256) the concept of model-view separation
 - It says model objects (like Sale) should not know about view (UI) objects
 - Idea is to maintain **low coupling** between model and UI layers



The Observer (Publish-Subscribe) Pattern III

- A "good" solution: an "observer interface" is defined
 - Define currTotalWindow to have an "onEvent()" method
 - When currTotalWindow is constructed, give its constructor (thereby enabling it to have) the Sale object as an attribute



The Observer (Publish-Subscribe) Pattern IV

- Sale contains an "addEventListener()" method
- Whenever the state of Sale changes significantly, it notifies anything that has called addEventListener() of the change
- The UI is *subscribing* to Sale
- Sale is *publishing* the "news"



The Observer (Publish-Subscribe) Pattern V

- The Sale object does not know about the currTotalWindow object
- It publishes its data to its subscriber list
- ...but does not know ahead of time who is on that list
- There is information hiding, indirection, and low coupling involved here
 - (how?)