

Behavioural Design Patterns

Part 3

Behavioural Design Patterns

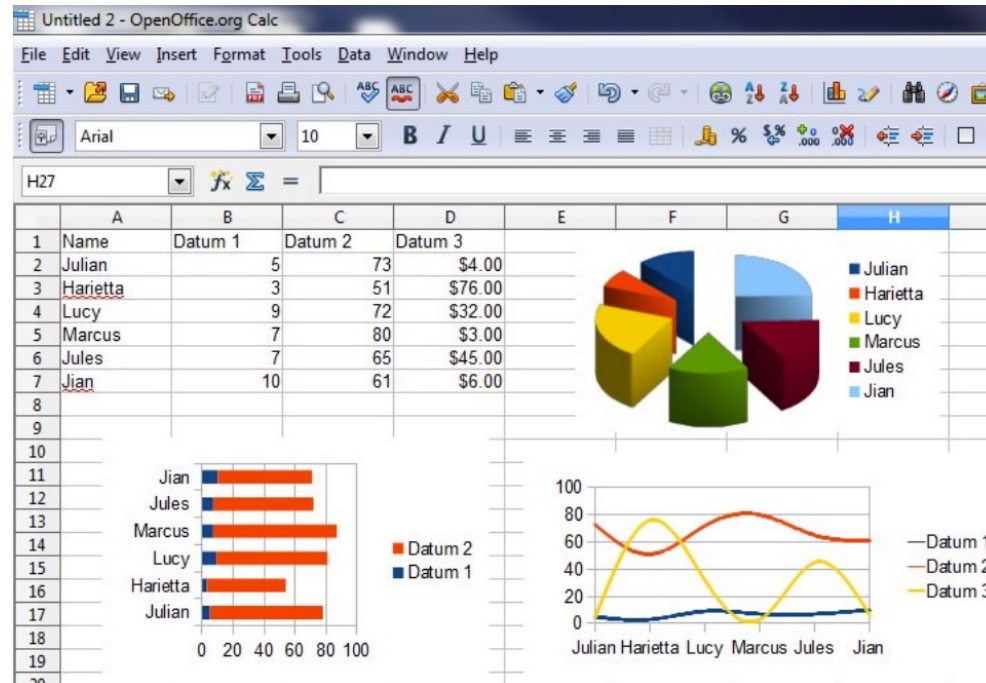
- State
- Strategy
- Observer
- Command
- Visitor



Behavioural Patterns: Observer

- We often have need to notify multiple subscribers about an event that occurs
 - We don't necessarily know which subscribers may be interested in our events
 - We might want to modify the subscriber list at run time
- Example: spreadsheet application with multiple graphs
 - Need to update graphs when spreadsheet data changes
 - Graphs can be added/removed at any time

Behavioural Patterns: Observer



Behavioural Patterns: Observer

Design Pattern:

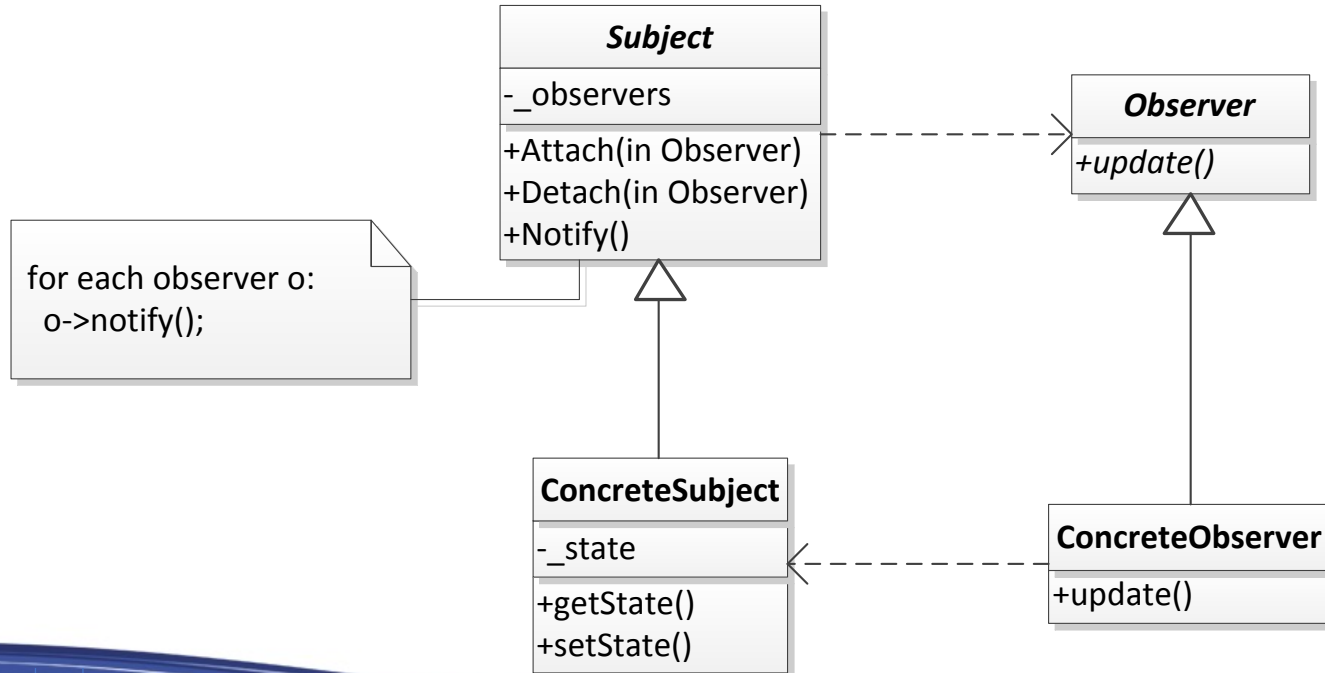
Observer

Defines a one-to-many dependency between objects so that when one object changes state, all of its dependents are notified and updated automatically.

Behavioural Patterns: Observer

- Applicability:
 - When an abstraction has two aspects, one dependent on the other; encapsulating these aspects in separate objects lets you vary and reuse them independently
 - When a change to one object requires changing others, and you don't know how many objects need to be changed
 - When an object should be able to notify other objects without making assumptions about who these objects are (i.e. we don't want these objects tightly coupled)

Behavioural Patterns: Observer



Behavioural Patterns: Observer

- Spreadsheet example:
 - Subject: Spreadsheet
 - ConcreteObserver: the various graphs
 - Pie, Bar, Line
 - When created, they are attached to the spreadsheet
 - Implementation of `update` is up to the individual graph classes
 - We could modify `update` to accept parameters passed from the spreadsheet
 - We could query the spreadsheet for the data we need

Behavioural Patterns: Observer

- Some Observers may observe more than one subject
 - We often pass in an event object containing details on which object generated the event as well as other pertinent information
- Deleting a Subject should cause Observers to remove any references to the Subject
 - Destructor of Subject can notify Observers of its deletion

Behavioural Patterns: Observer

- Consequences:
 - Abstract coupling between Subject and Observer; a Subject only knows that it has a list of Observers – it doesn't know anything about them
 - Support for broadcast communication (one-to-many)
 - Unexpected updates
 - Observers have no knowledge of each other's presence
 - A seemingly innocuous operation on a Subject may cause a cascade of updates to Observers and their dependent objects