XIV. Design Patterns

Acknowledgment: these slides are based on Prof. John Mylopoulos slides which are used to teach a similar course in the University of Toronto – St. George campus. Used with Permission.

Command pattern Applicability

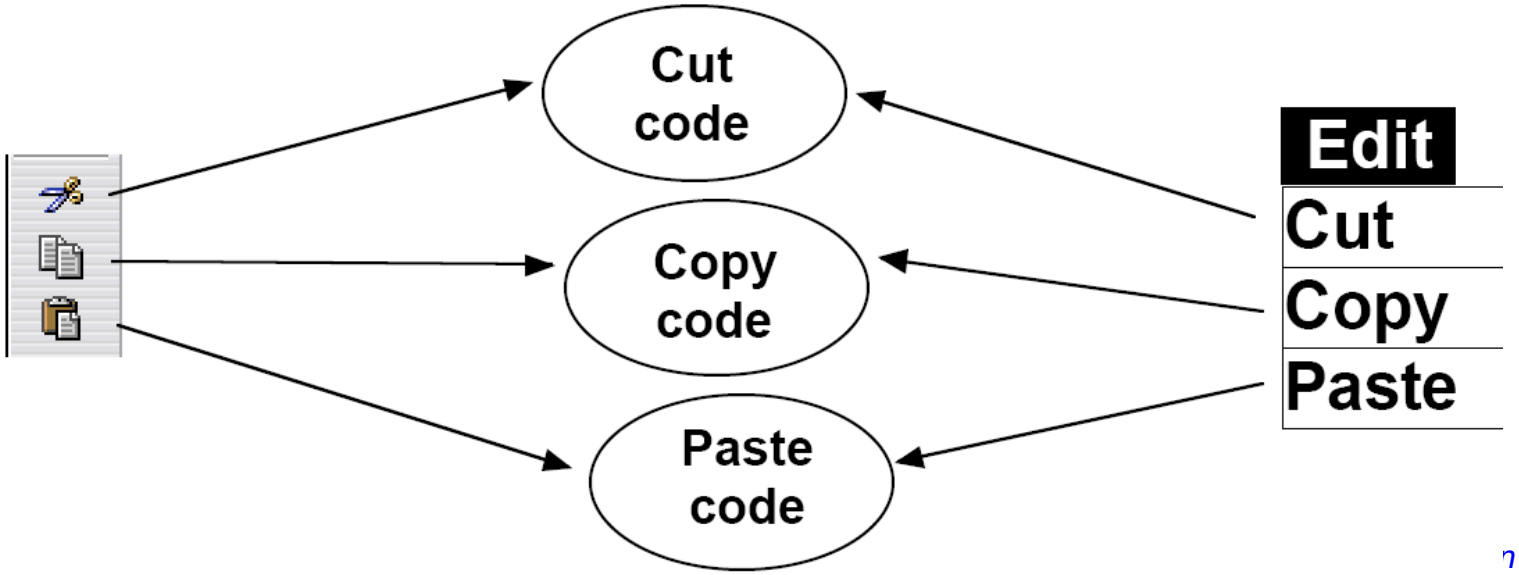
"Encapsulate a request as an object, thereby letting you

- parameterize clients with different requests,
- queue or log requests, and
- support undoable operations."

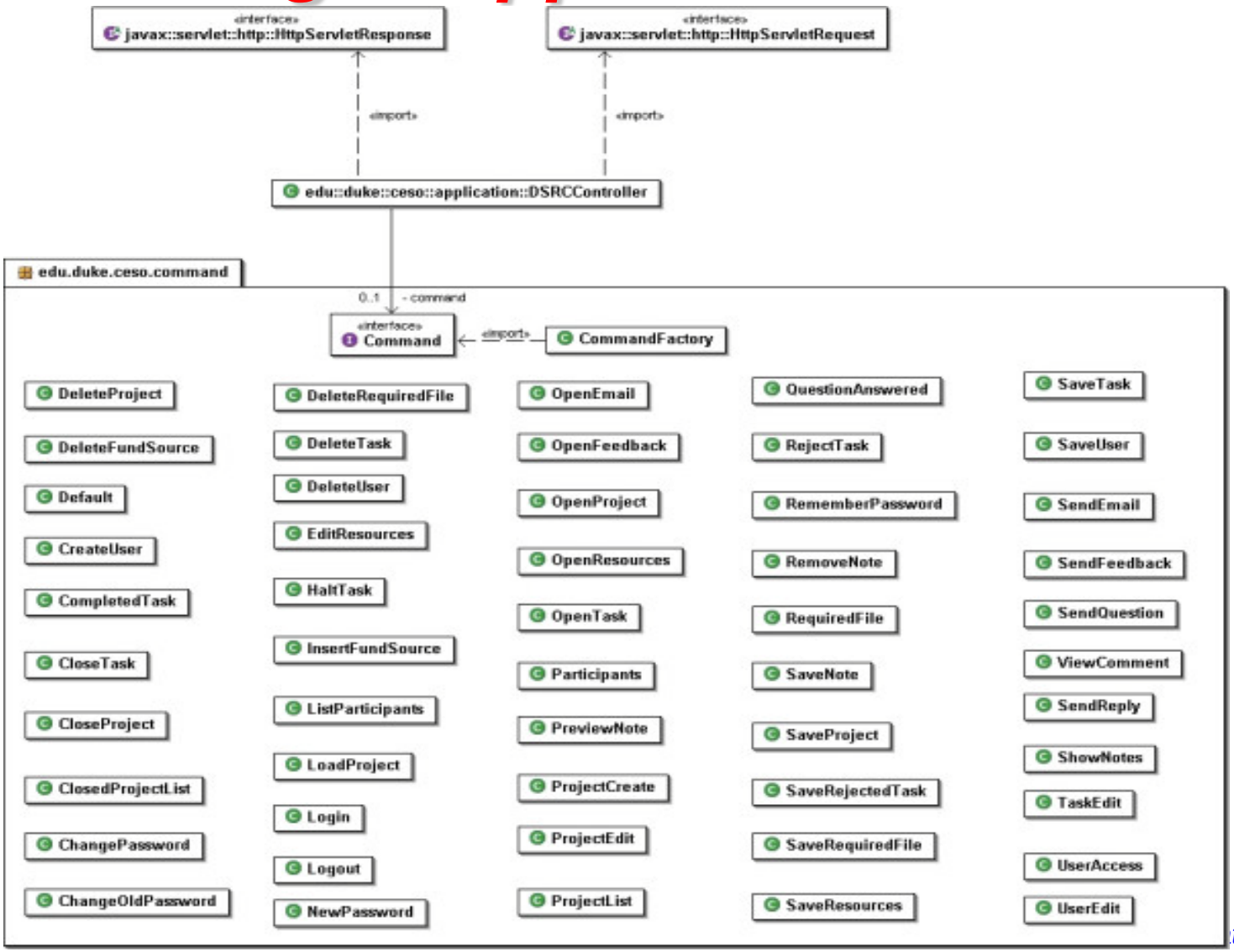
- **Uses:**
  - Undo queues, can add now since each command is sent through a command object and we can create a history of commands within this object
  - Database transaction buffering
  - Structure the application around commands

Common UI commands

- **it is common in a GUI to have several ways to activate the same behavior**
  - example: toolbar "Cut" button and "Edit / Cut" menu
  - this is *good* ; it makes the program flexible for the user
  - we'd like to make sure the code implementing these common commands is not duplicated



Command pattern facilities understanding of applications code

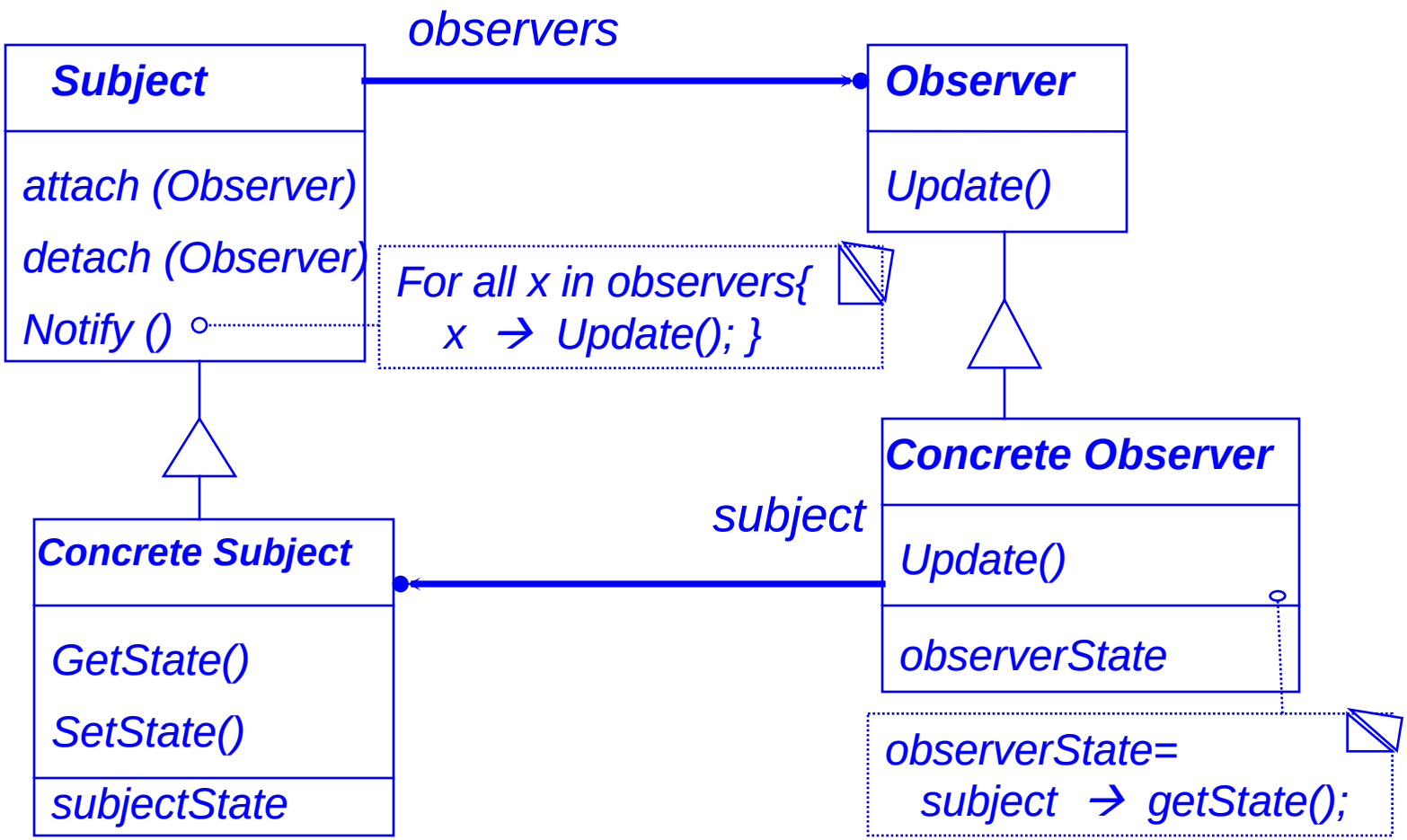


Observer pattern

- **"Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically."**
- **Also called "Publish and Subscribe"**

- **Uses:**
  - Maintaining consistency across redundant state
  - Optimizing batch changes to maintain consistency

Observer Pattern



Typical command line program

- **Non-interactive**
- **Linear execution**

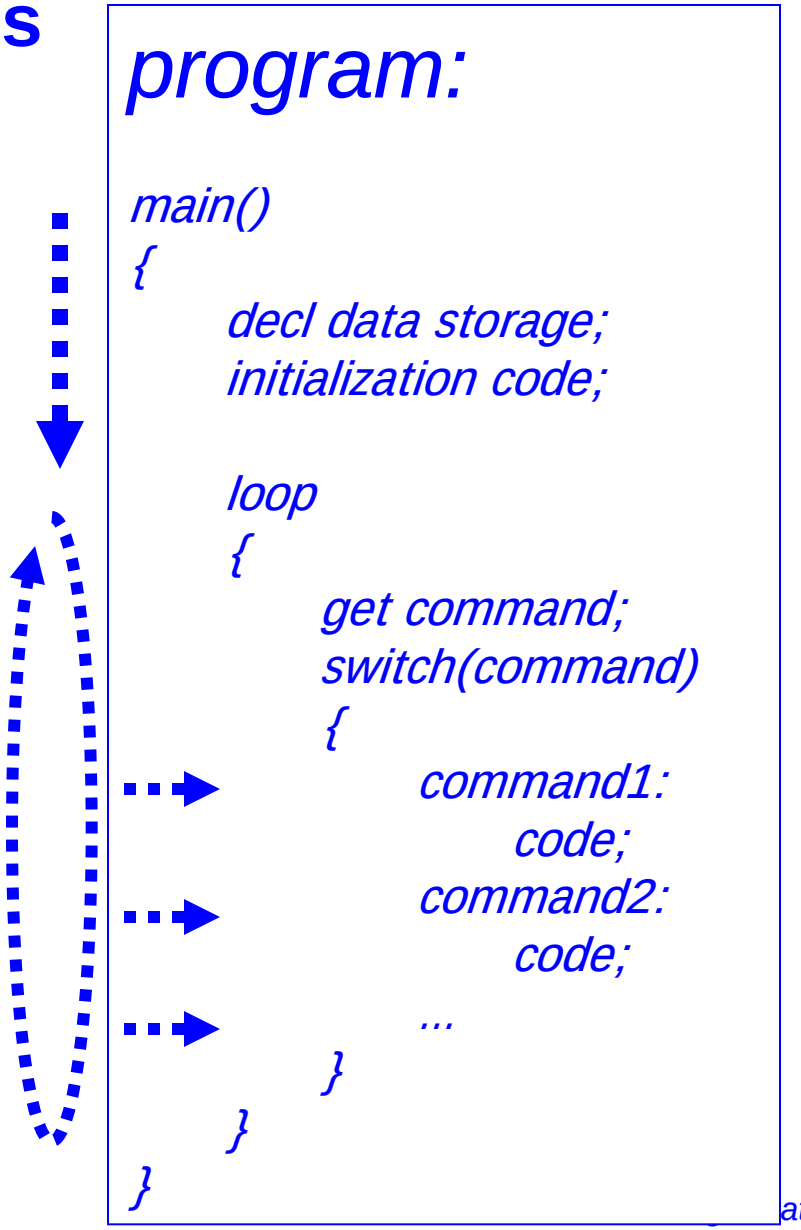
IBM 705

Univac 1956

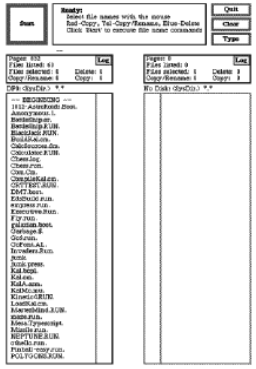
```
program:

main()
{
        code;
        code;
        code;
        code;
        code;
        code;
        code;
        code;
        code;
        code;
        code;
        code;
}
```

Interactive command line program

- User input commands
- Non-linear execution
- Unpredictable order
- Much idle time

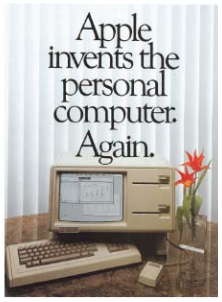


Interactive Graphical User Interface

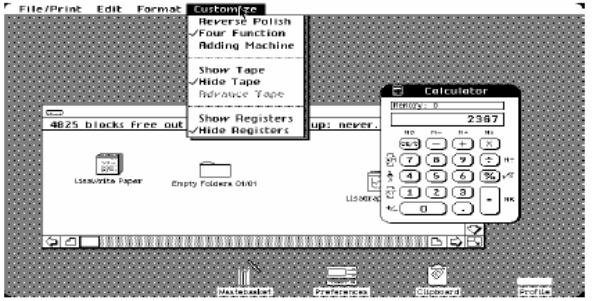


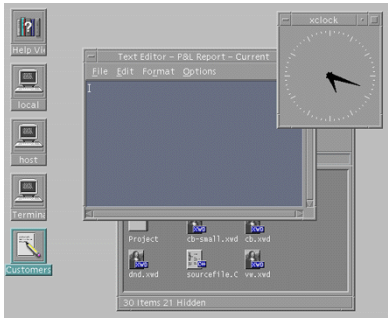
Xerox Alto

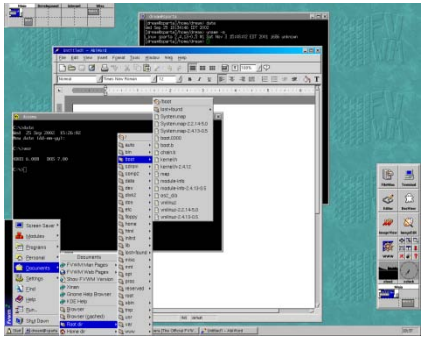
Xerox PARC, 1973



Lisa Interface

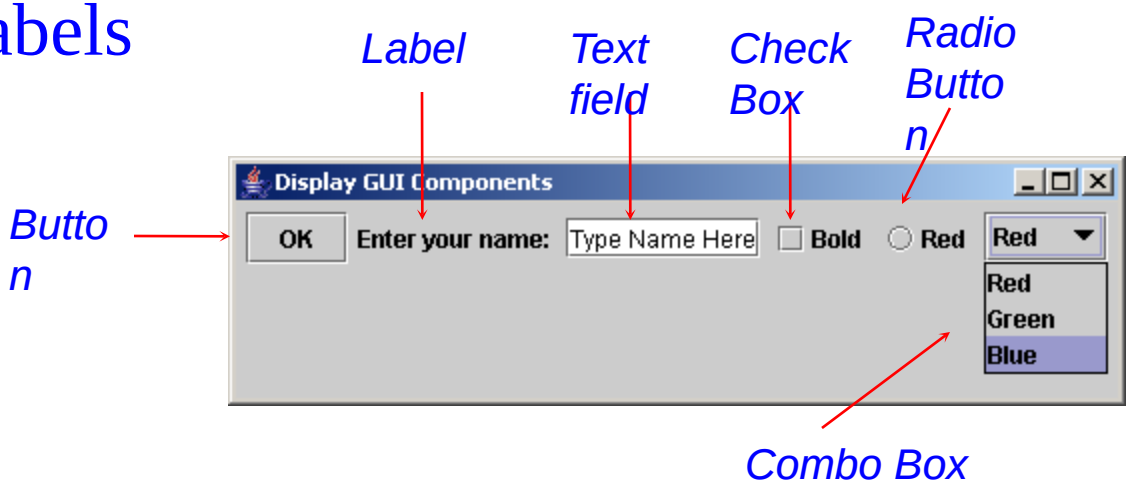






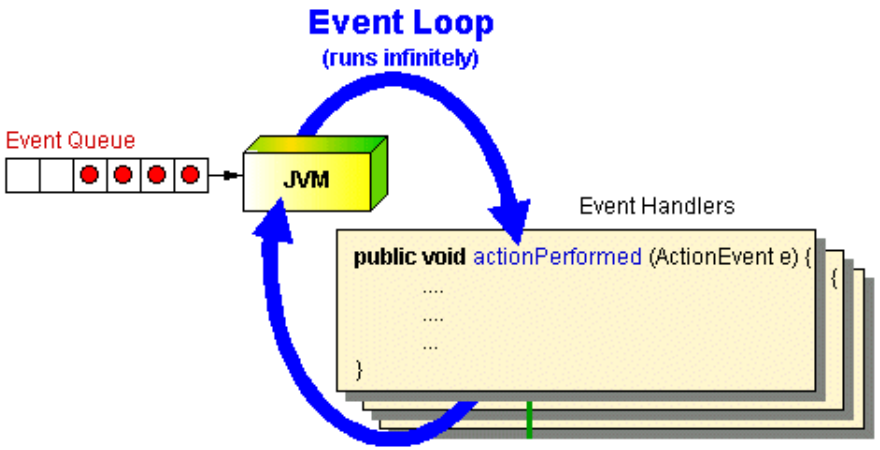
Interactive Graphical User Interface

- **What's make a GUI GUI?**
  - Windows
  - Selection controls: drop-downs, radio-buttons, check boxes, menus,..
  - Activation controls: buttons, icons
  - Input controls: text fields, text areas
  - Structure information visually: lists, grids, trees, labels



Java GUI program

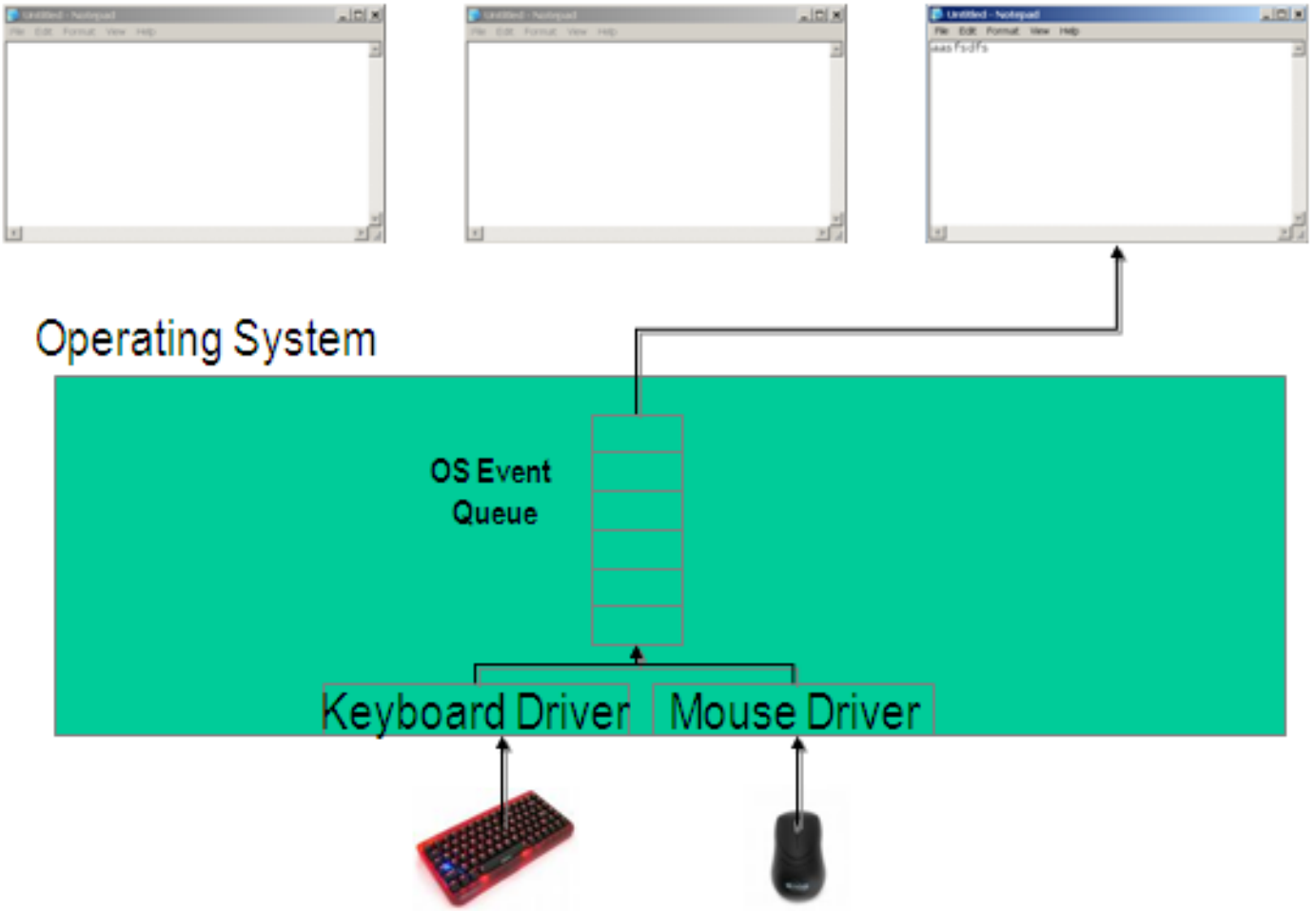
- **Event loop automatic in separate program**



Java program:

```
Class{
main()
{
        decl data storage;
        initialization code;

        create GUI objects;
                register listeners;
}

listener1()
{       do stuff;
}
listener2()
{       do stuff;
}
...
```

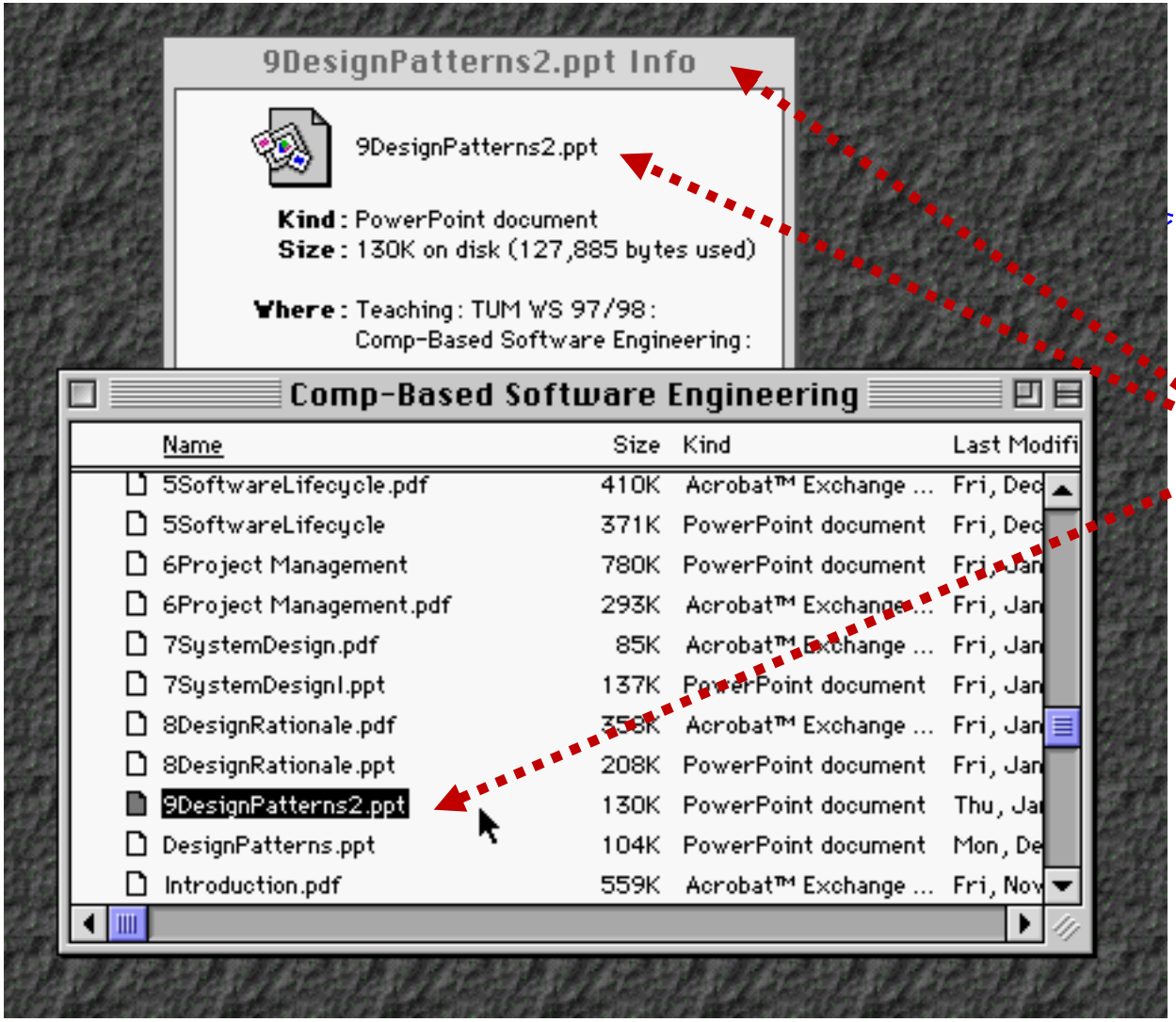
Input Events



Observer pattern (continued)

Observers

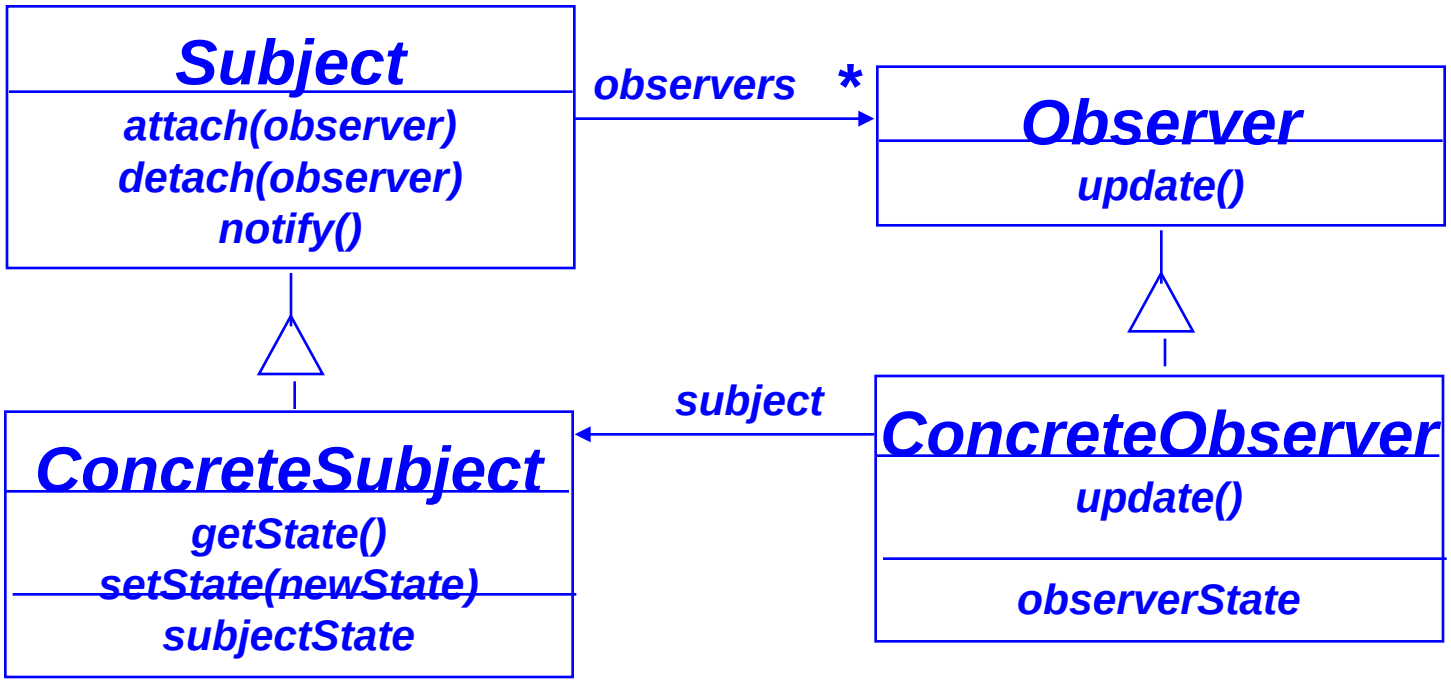
Subject



9DesignPatterns2.pp

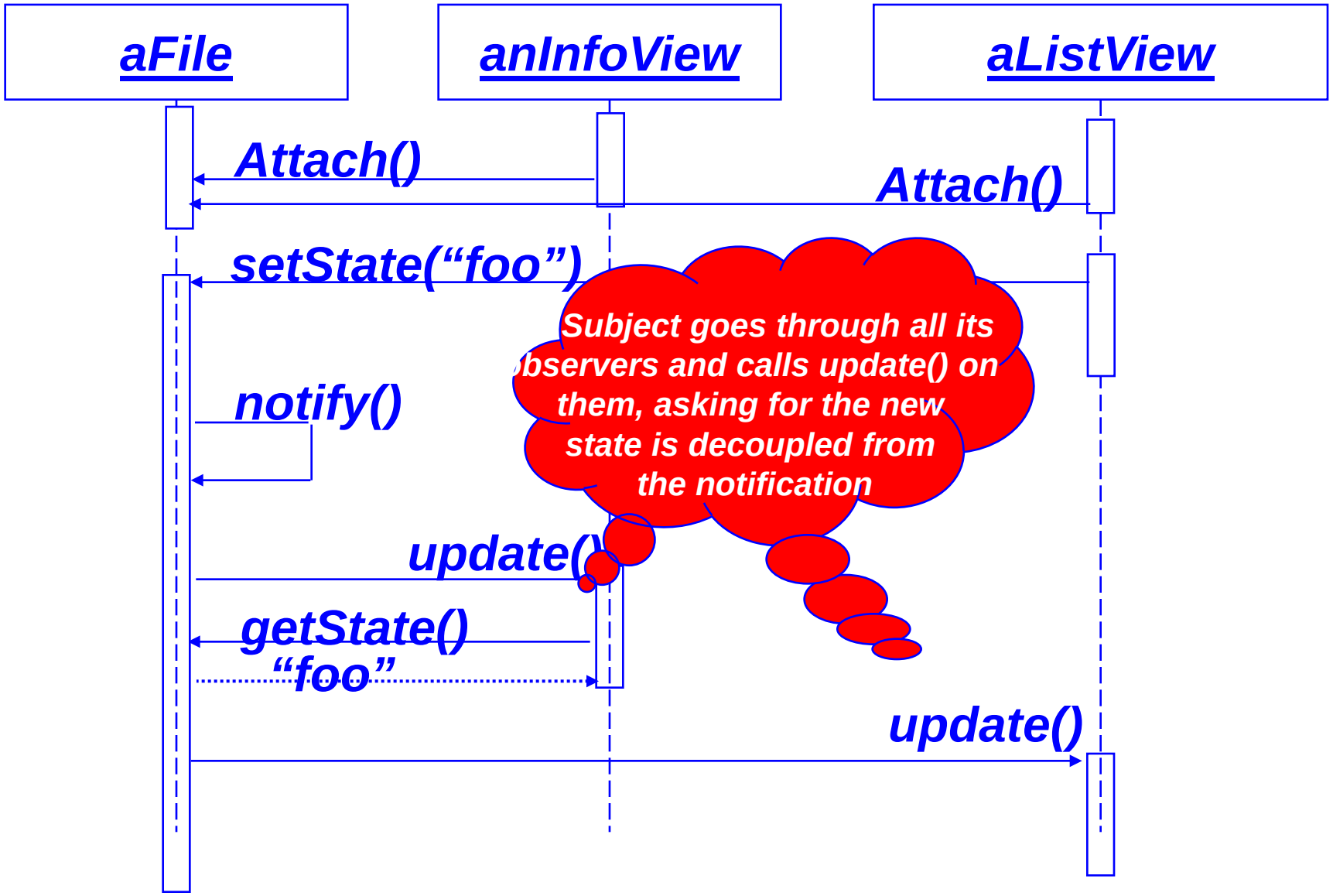
Change name to F

Observer pattern (cont'd)



- The Subject represents the actual state, the Observers represent different views of the state.
- Observer can be implemented as a Java interface.
- Subject is a super class (needs to store the observers vector) *not* an interface.

Sequence diagram for scenario: Change filename to “foo”



Observer Pattern: Consequences

- ***Abstract coupling*** **between subject and observer. Subject has no knowledge of concrete observer classes. (What design principle is used?)**
- *Support for broadcast communication. A subject need not specify the receivers; all interested objects receive the notification.*
- *Unexpected updates: Observers need not be concerned about when then updates are to occur. They are not concerned about each other's presence. In some cases this may lead to unwanted updates.*