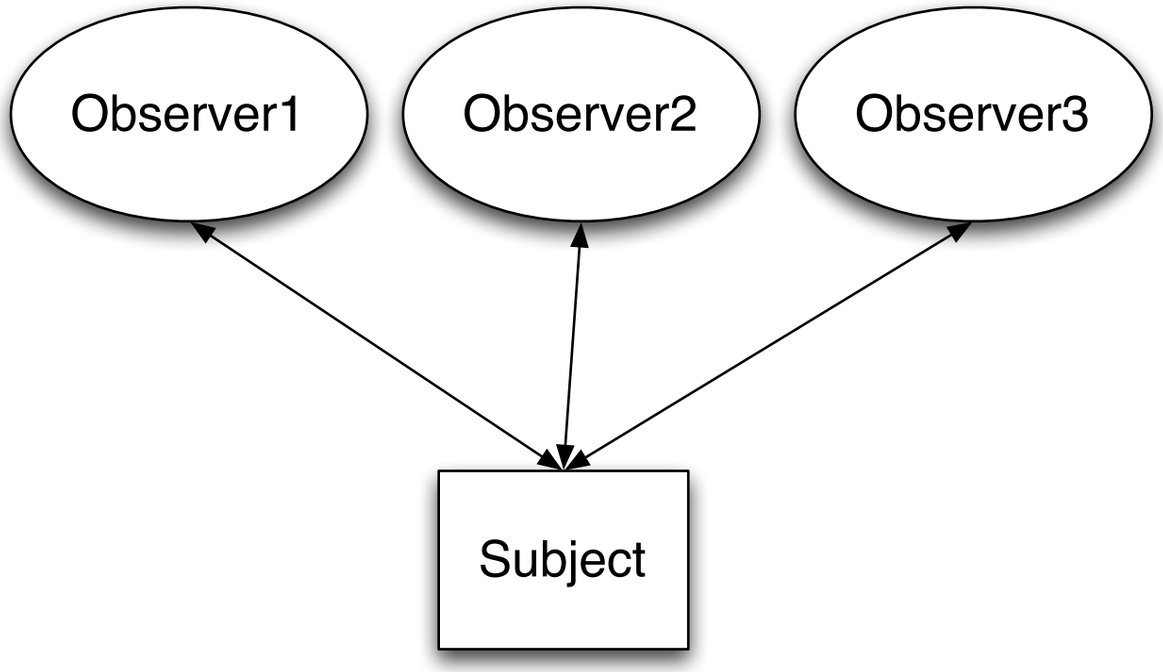
CS 535 Object-Oriented Programming & Design
Fall Semester, 2008
Doc 15 Some patterns
Nov 2010

References

Design Patterns, Gamma, Helm, Johnson, Vlissides, 1995

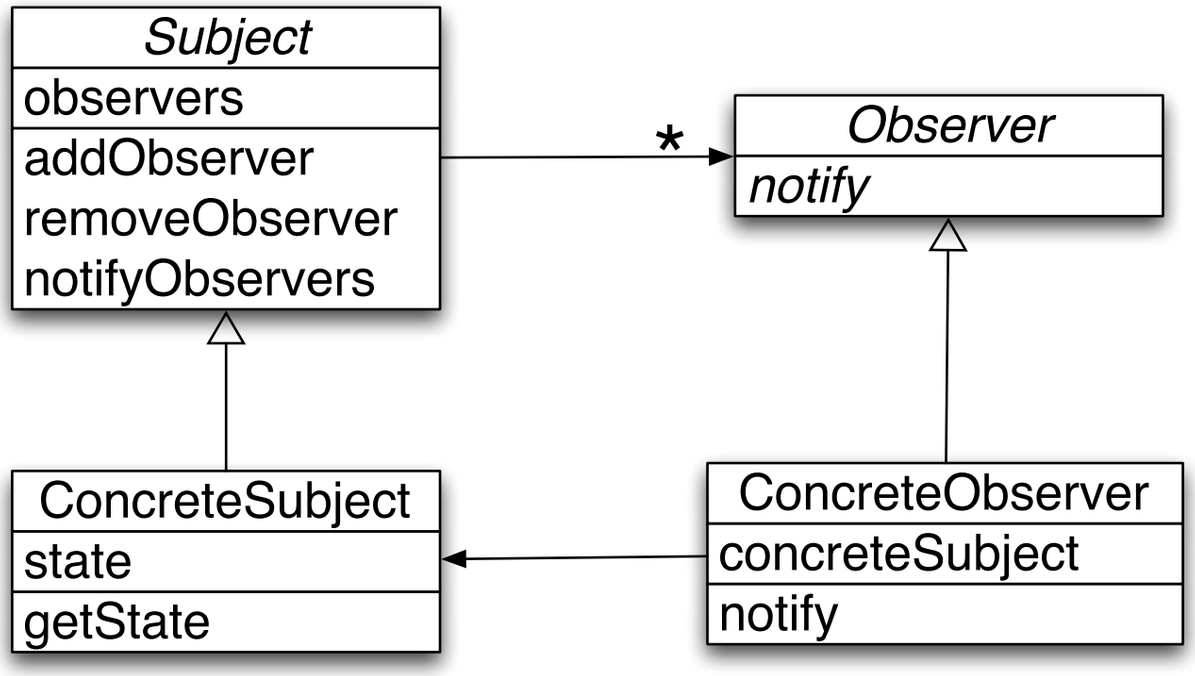
Object-Oriented Design Heuristics, Riel, 1996

Observer Pattern



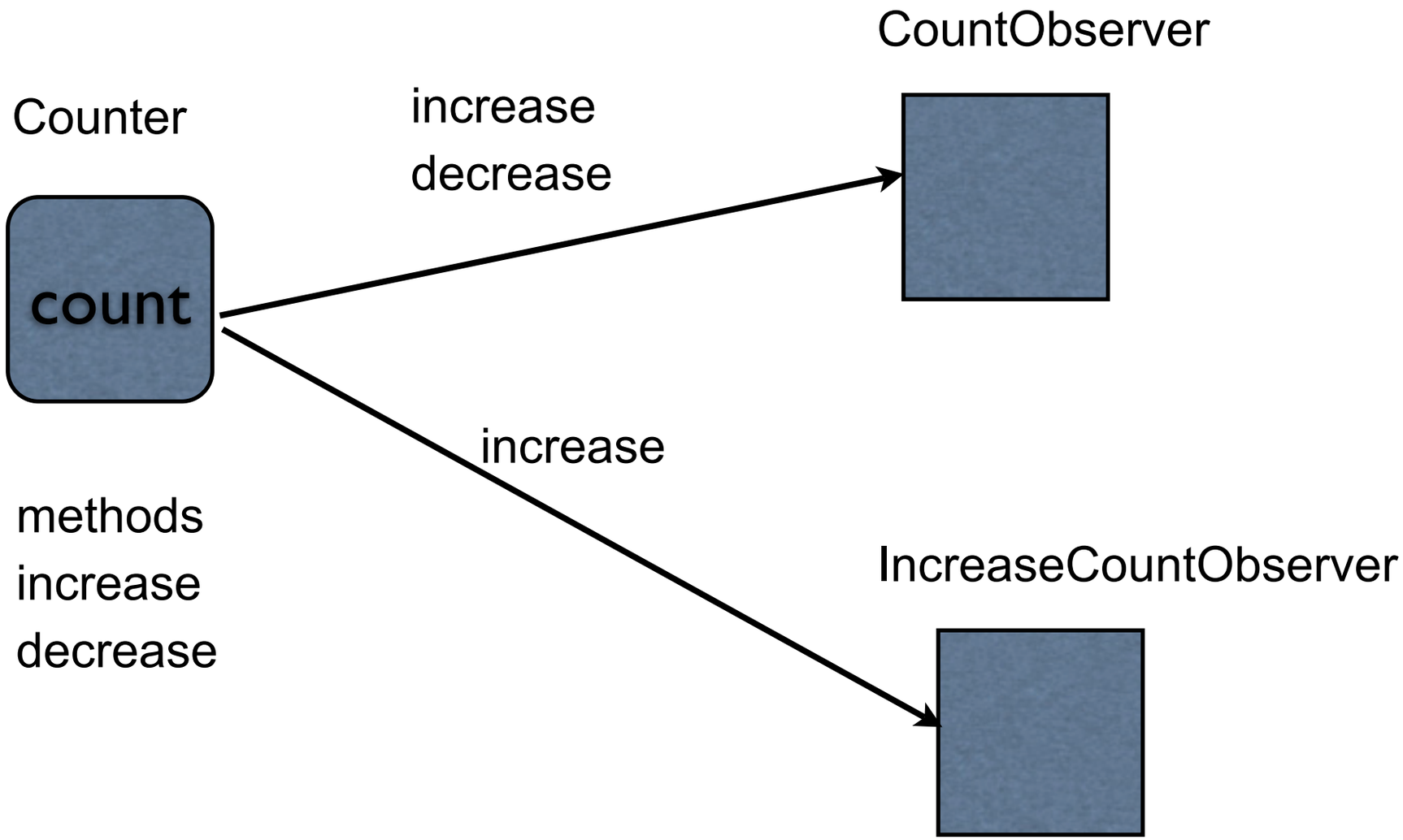
Subject notifies all observers when it changes

Keeping it Flexible




```
Subject>>notifyObservers
    observers do: [:each | each notify]
```

Example: Counter



Counter Class

instance variables

count

instance methods

decrease

```
count := count - 1.
```

increase

```
count := count + 1.
```

count

```
^count
```

Counter & Dependents without Observer

instance variables

count

countObserver

increaseObserver

instance methods

```
decrease
	count := count - 1.
	countObserver newCount: count.

increase
	count := count + 1.
	countObserver newCount: count.
	increaseObserver countIncreased.

count
	^count
```

Comment

Counter Class knows:

- Class of Views
- Number of views
- Calls specific methods in views

Add new views/dependents requires

- Adding more instance variables
- Modifying methods

Counter & Dependents with Observer

instance variables

count

dependents (collection)

instance methods

```
decrease
	count := count - 1.
	self changed: #decrease

increase
	count := count + 1.
	self changed: #increase

changed: aSymbol
	dependents do: [:each | each update: aSymbol].

addDependent: anObject
	dependents add: anObject

count
	^count
```

Observers update:

Implement update: method

Called when subject has changed

updates observers state

Example

```
counter := Counter new.
simpleObserver := CountObserver observe: counter.
counter increase.
increase := IncreaseObserver observe: counter.
```

```
CountObserver>>update: aSymbol

    Transcript
        show: 'Change';
        space;
        show: subject count printString;
        cr
```

Comments

Special model class

implements changed:, addDependent:

contains dependents instance variable

View/dependents

implement update: method

Counter class

any number observers

Observers can be any type

Observers know about Counter class

Comments

Observer pattern part of many class libraries

- Java, Smalltalk

Smalltalk

- Object class implements much of the pattern
- More options that shown

Coupling

Measure of the interdependence among modules

"Unnecessary object coupling needlessly decreases the reusability of the coupled objects "

"Unnecessary object coupling also increases the chances of system corruption when changes are made to one or more of the coupled objects"

Linked List Example

```
Node>>print
    Transcript
        show: value printString.

LinkedList>>print
    Transcript show: 'List('.
    self
        do: [:each | each print ]
        separatedBy: [Transcript show: ', ']
```

Linked list coupled to Transcript

Cant use it in

- GUI
- Network code
- Background process

Linked List Example

```
Node>>print: aStream
    aStream
        nextPutAll: value printString.

LinkedList>>print: aStream
    aStream nextPutAll: 'List('.
    self
        do: [:each | each print: aStream ]
        separatedBy: [aStream nextPutAll: ', ']
```

Linked list coupled to Stream interface

Can use it with

- Transcript
- Files
- NetworkStream

With some work can extract data

Linked List Example

```
LinkedList>>asOrderedCollection
    |collection |
    collection := OrderedCollection new.
    self
        do: [:each | collection add: each value ]
    ^collection
```

Linked list coupled to OrderedCollection

Can write code to put data

- Transcript
- Files
- Any stream
- GUI

Easy access to data

GUIs & Coupling

Domain information
- Customer records
- Inventory
- Names
- Reports
- Addresses

Application/GUI information
- Menus
- Error Messages
- Help information
- Labels

Keep domain and application information separate

Application information changes faster

Often there is multiple view of domain information

Heuristic 3.5

In application that consists of an object-oriented model interacting with a user interface,

the model should never be dependent on the interface

the interface should be dependent on the model

Model provides accessors to data that interface needs

Model-View-Controller (MVC)

Model

Encapsulates

Domain information
Core data and functionality

Independent of

Specific output representations
Input behavior

View

Display data to the user

Obtains data from the model

Multiple views of the model are possible

Controller

Handles input

- Mouse movements and clicks
- Keyboard events

Each view has it's own controller

Programmers commonly don't see controllers

MVC & Coupling

Model should know a little as possible about views

Views tend to know a lot about model

View is observer

Model is subject

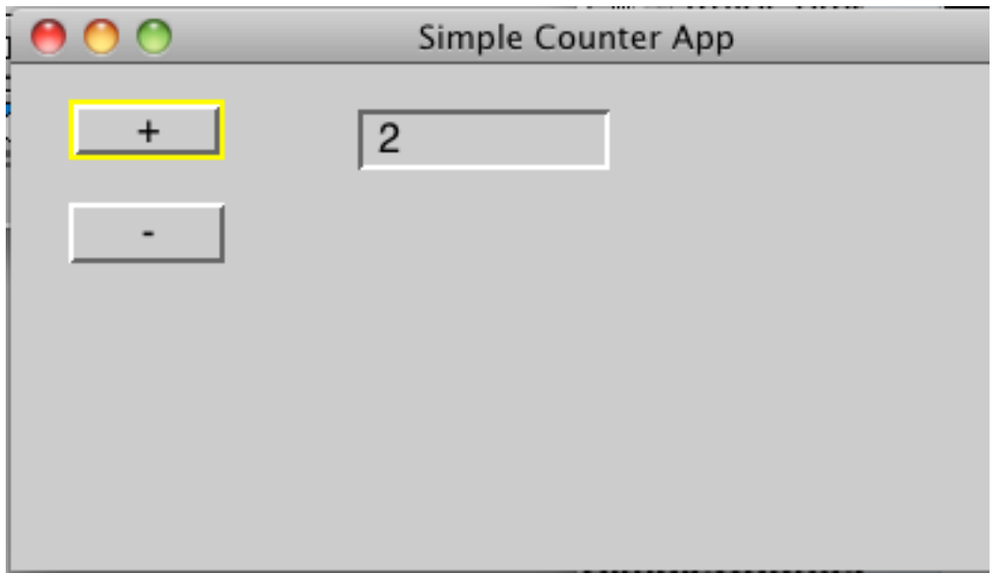
ValueHolder

```
'cat' asValue

10 asValue
```

```
Object>>asValue
	"Return a ValueHolder on the receiver"

	^ValueHolder with: self
```

ValueHolder is a subject

Simple Counter App



SimpleCounterApp

Parent class ApplicationModel

instance variable: count

```
increase
        count value: (count value + 1)

decrease
        count value: (count value - 1)
```

```
count
        ^count isNil
                ifTrue:
                        [count := 0 asValue]
                ifFalse:
                        [count]
```

Problem

Textfield Widget works with ValueHolder object

It changes the value of the ValueHolder

What if we want more complex subject?

How to use Counter Class in App

Not a simple value for ValueHolder

Counter Class

instance variables

count

instance methods

```
decrease
        count := count - 1.

increase
        count := count + 1.

count: aNumber
      count := aNumber

count
        ^count
```

Adapters

Basic Idea

TextField

TextField uses

x value: aNumber

x value

Adapter maps

Counter class has

decrease
    count := count - 1.

increase
    count := count + 1.

count: aNumber
    count := aNumber

count
    ^count

SimpleCounterApp

```
initialize
	count := Counter new

decrease
	count decrease

increase
	count increase
```

```
count
	| countAdapter |
	countAdapter := AspectAdaptor subject: count.
	countAdapter
		forAspect: #count;
		subjectSendsUpdates: true.
	^countAdapter
```

Counter Class

instance variables

count

instance methods

```
decrease
        count := count - 1.
        self changed: #count

increase
        count := count + 1.
        self changed: #count

count: aNumber
      count := aNumber

count
        ^count

initialize
        count := 0
```

How it Works

Count is subject for AspectAdapter

AspectAdapter observes Count

AspectAdapter is subject for TextField

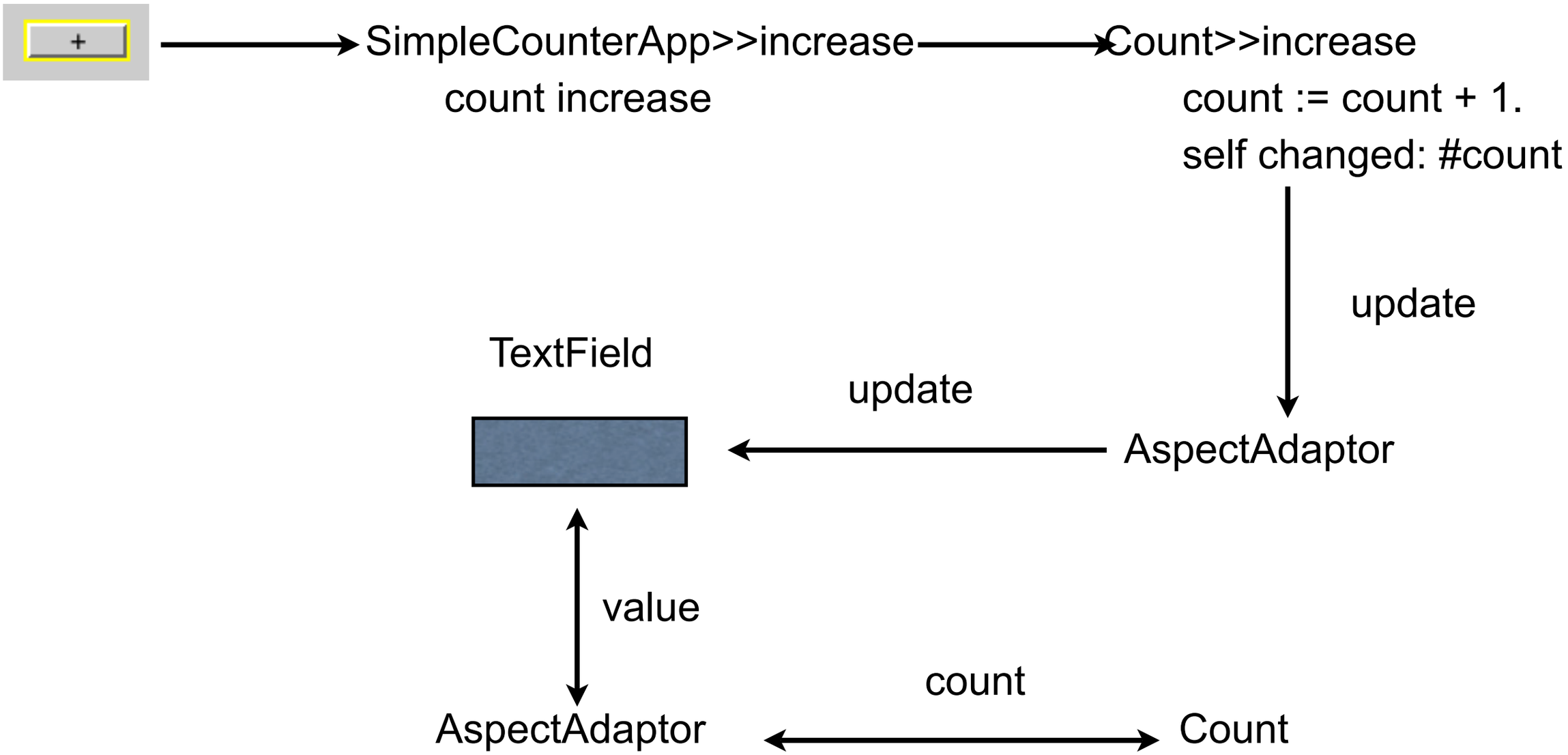
TextField observes AspectApapter

How it Works

User edits text

How it Works

User clicks on + button



Singleton

Ensure a class only has one instance

Provide global point of access to single instance

Smalltalk Implementation

Class instance variable

instance

Class methods

new

```
self error: 'Singleton class, use instance for instance of class'
```

instance

```
instance ifNil: [instance := super new initialize].
^instance
```