Design Patterns

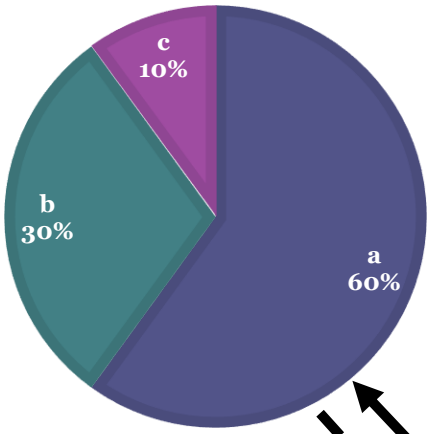
Observer and Command

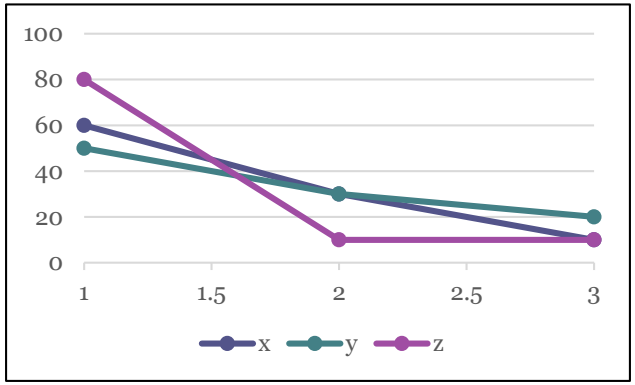
Dr. Chad Williams
Central Connecticut State University

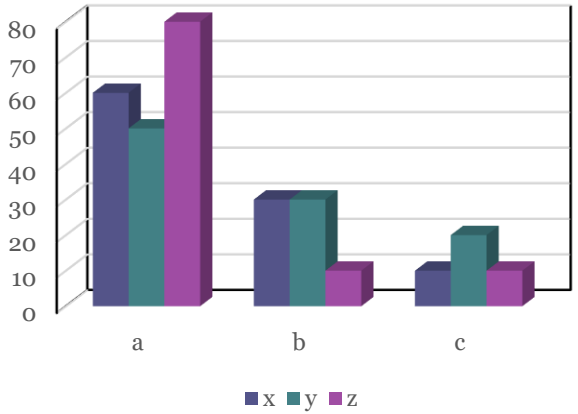
Design pattern: Observer

- **Category**: Behavioral design pattern
- **Intent**:
  - Define one-to-many dependency so that when one object changes state all of its dependents are notified and updated automatically
- **Motivation**
  - Collection of cooperating objects needing to maintain consistency/updates without becoming tightly coupled

Motivation cont.







…

	a	b	c
x	60	30	10
y	50	30	20
z	80	10	10

notification

modifications

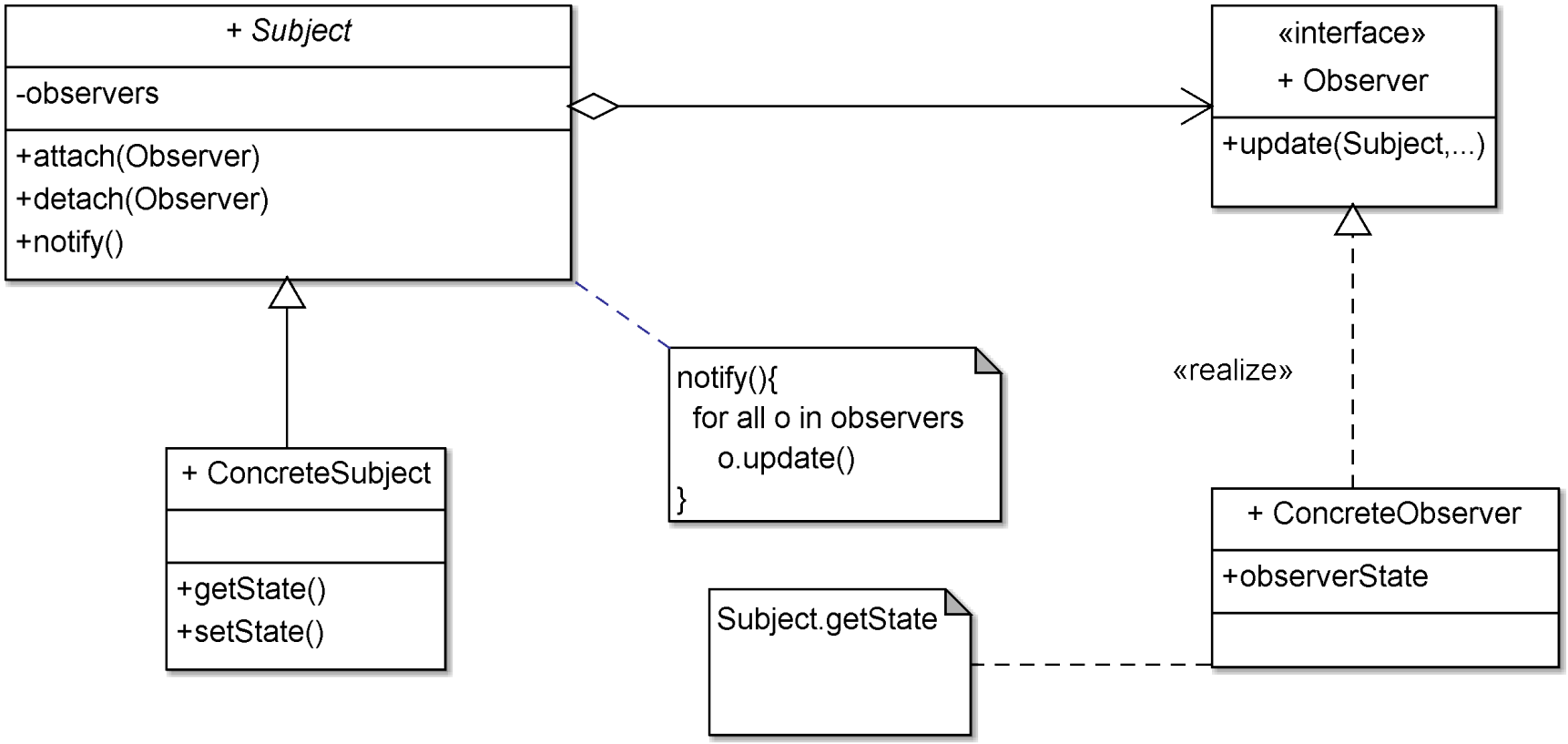
Applicability

- Abstraction has two aspects, one dependent on the other. Encapsulating in separate objects lets you vary and reuse them independently
- Changing one requires changing others and you don't know how many others will need to change
- Object should be able to notify others without making assumptions about that object – you don't want the objects tightly coupled

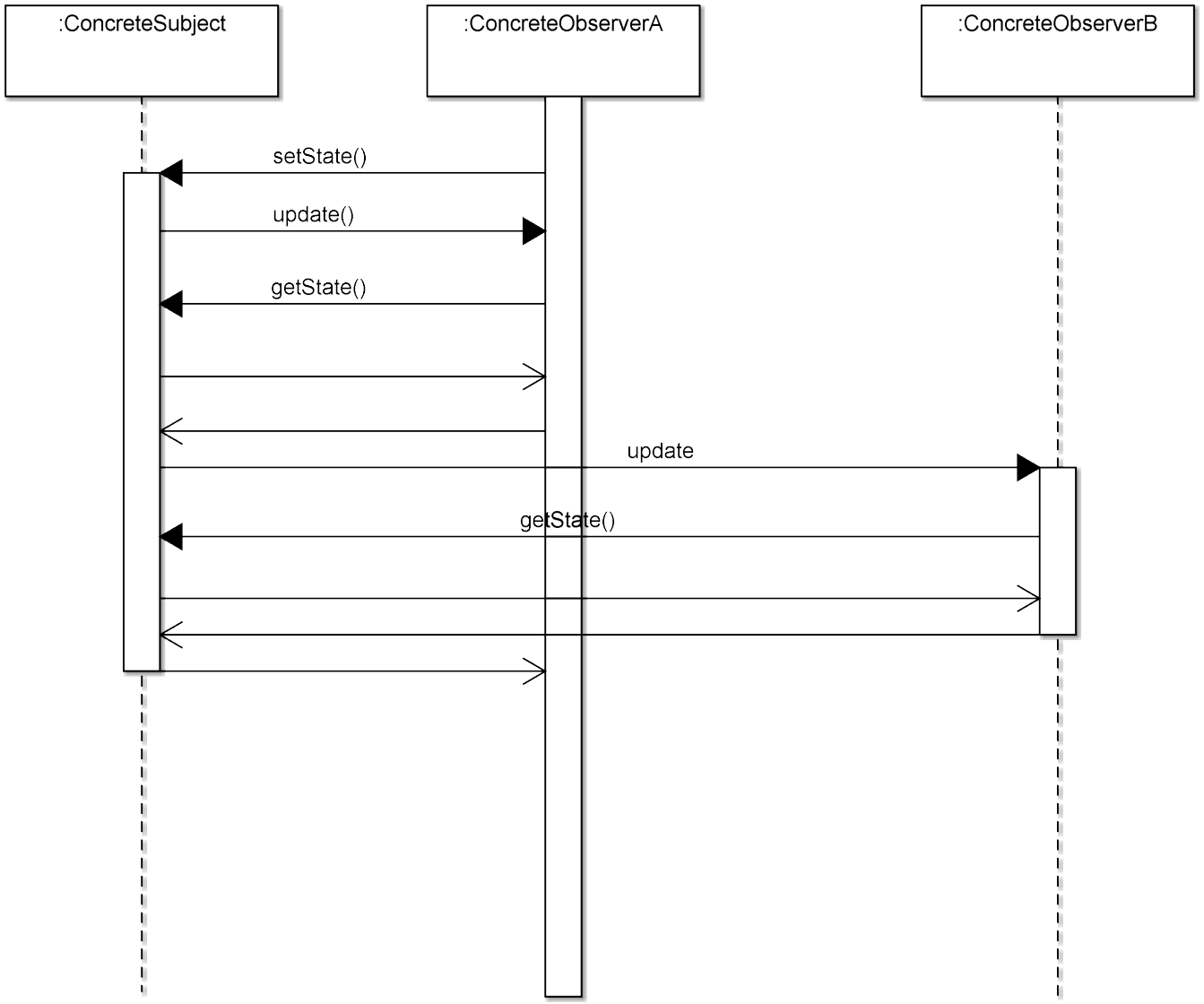
Participants

- Subject
  - Knows its observers. Any number of Observer objects may observe
  - Provides interface for attaching/detaching Observer object
- Observer
  - Defines an updating interface for objects that should be notified of changes in a subject
- ConcreteSubject
  - Stores state of interest to ConcreteObservers
  - Sends notification to its observers when its state changes
- ConcreteObserver
  - Maintains reference to ConcreteSubject object
  - Stores state that should stay consistent with the subject's
  - Implements the Observer updating interface

Observer UML



Sequence


:ConcreteSubject
:ConcreteObserverA
:ConcreteObserverB
setState()
update()
getState()
update
getState()

In class examples

- Excel document
- Class registration waitlist

Design pattern: Command

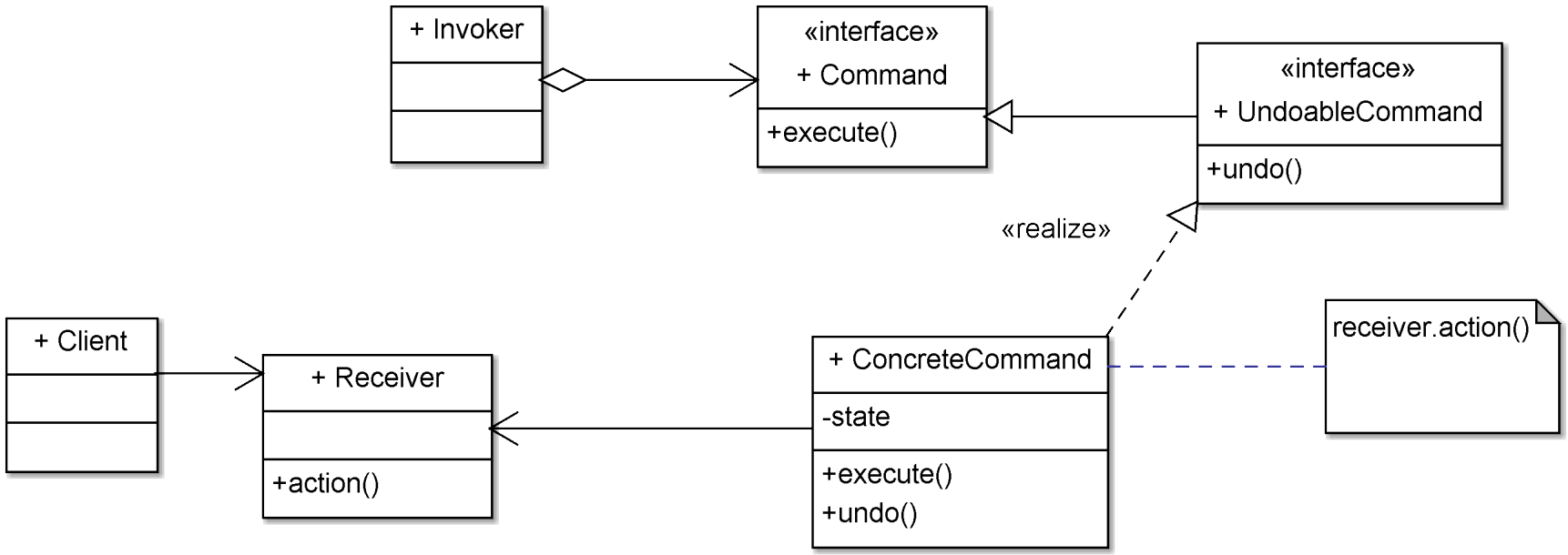
- **Category**: Behavioral design pattern
- **Intent**:
  - Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue, or log, and support undoable operations
- **Motivation**
  - Sometimes necessary to issue request to object without knowing anything about operation being requested or the receiver of the request
  - Example application menu framework same regardless of application, but depending on app options and actions change

Applicability

- Parameterize objects by an action to perform such as menu item
- Specify, queue, and execute requests at different times
- Support undo
- Log calls
- Encapsulate group of actions in a transaction

Participants

- Command
  - Declares an interface for executing an operation
- ConcreteCommand (PasteCommand, OpenCommand)
  - Defines a binding between a Receiver object and an action
  - Implements Execute by invoking the corresponding operation(s) on Receiver
- Client (Application)
  - Creates a ConcreteCommand object and sets its receiver
- Invoker (MenuItem)
  - Asks the command to carry out the request
- Receiver (Document, Application)
  - Knows how to perform the operations associated with carrying out the request


+ Invoker
«interface»
+ Command
+execute()
«interface»
+ UndoableCommand
+undo()
«realize»
+ Client
+ Receiver
+action()
+ ConcreteCommand
-state
+execute()
+undo()
receiver.action()

In class examples

- Calculator
- MS Word