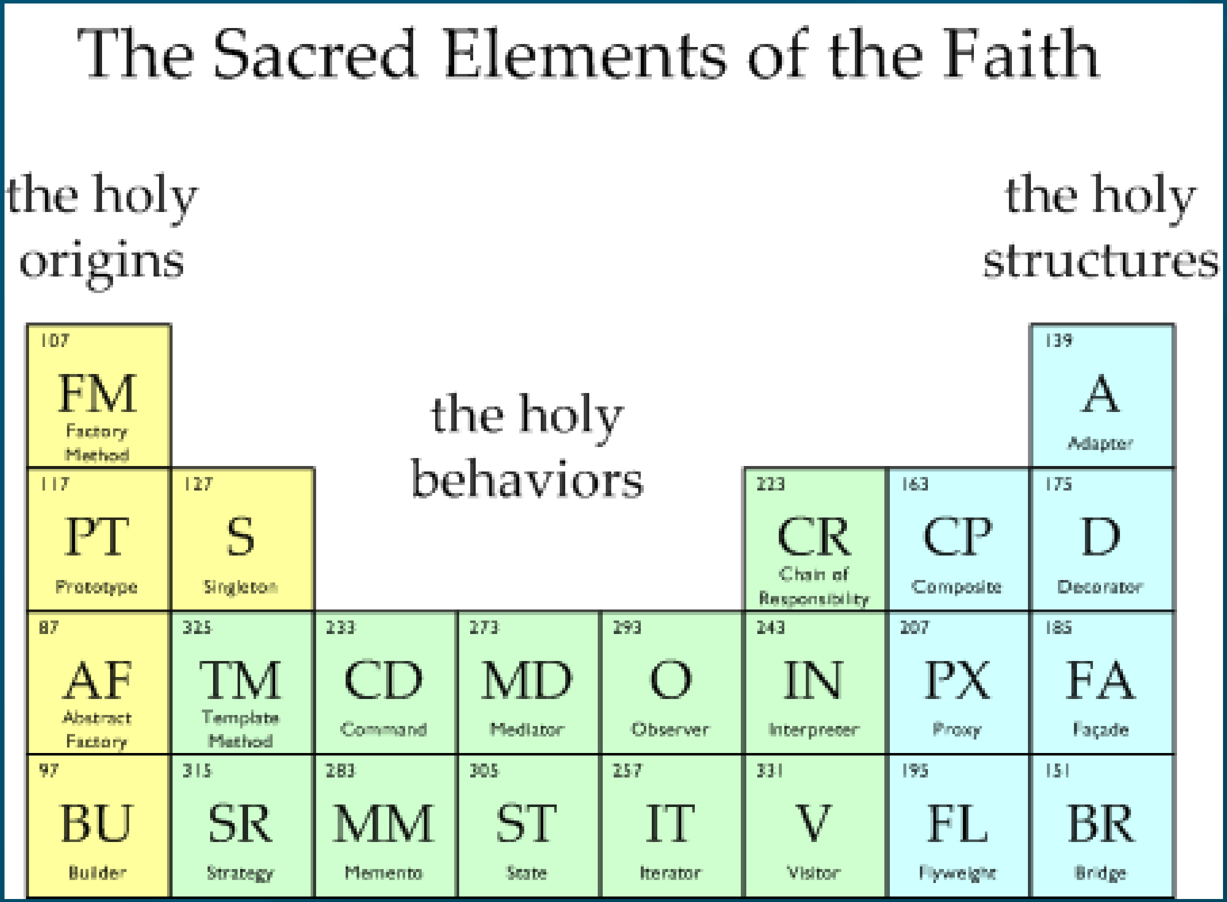
Design Patterns (2): Structural and Behavioural

5 April 2011
CMPT166
Sean Ho
Trinity Western University

See also:
Vince Huston,
JavaCamp,
OODesign.com

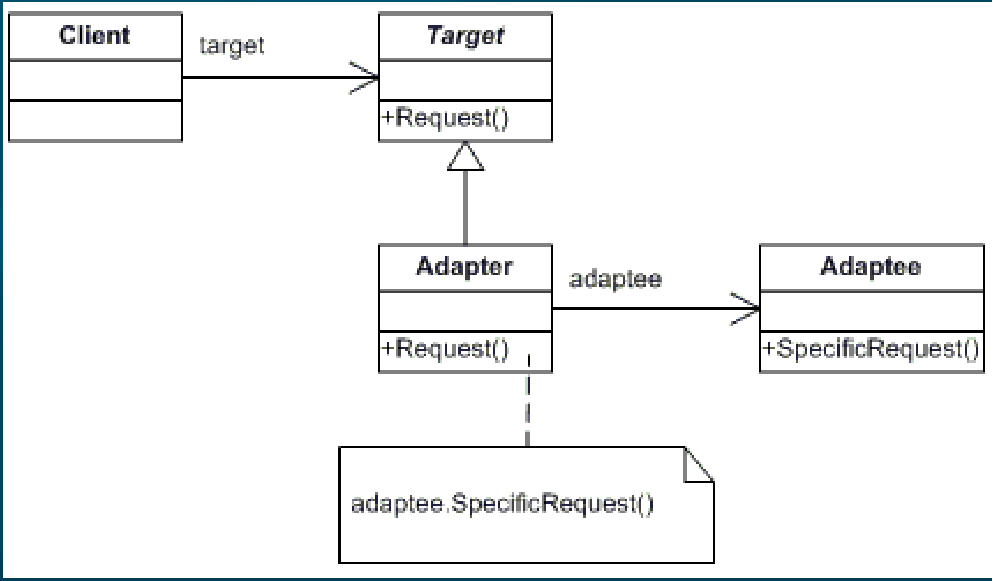
Classes of patterns (GoF)

- Creational
  - Interfaces to generate new objects
- Structural
  - How to organize a large system in components
- Behavioural
  - How components interact with each other to accomplish a common goal



Structural: Adapter

- Convert interface of a class so two incompatible classes can work together
- Like converting 3-prong plug to 2-prong socket, or impedance matching electrical signals
- e.g., integrating prepackaged software with your existing system



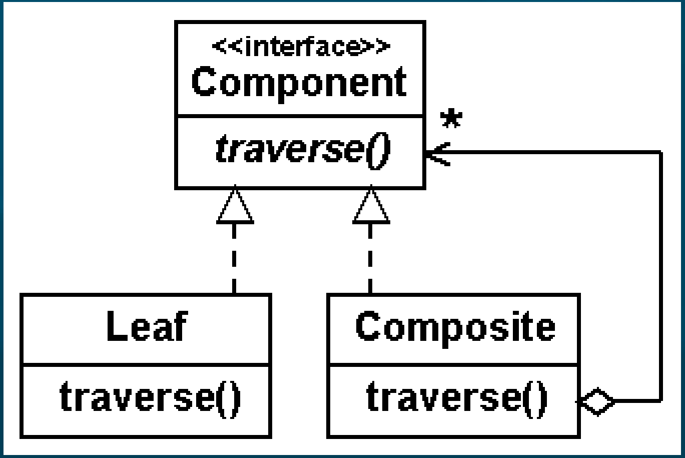
Structural: Bridge

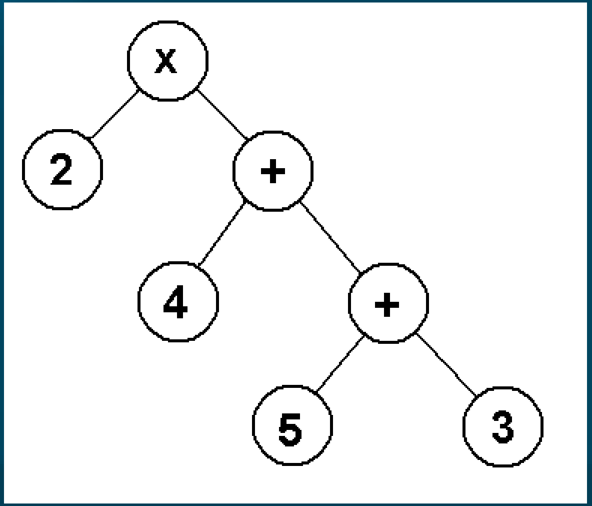
- Decouple an abstraction from its implementation so that the two can vary independently
- e.g., light switch abstract concept vs. implementation of kinds of switches

$$\begin{tikzcd}
\text{client} \arrow[r] & \text{abstraction} \arrow[r] & \text{implementor} \\
 & \text{refinement} \arrow[u] & \text{refinement} \arrow[u]
\end{tikzcd}$$

Structural: Composite

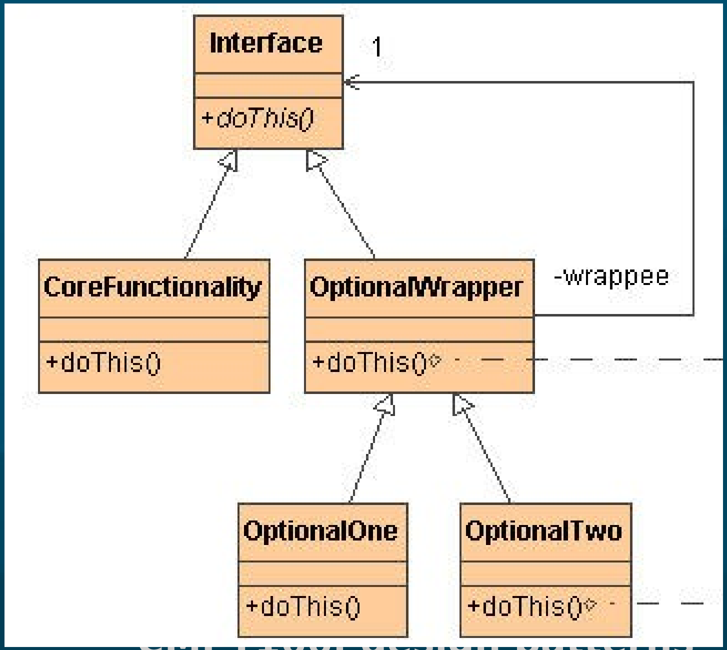
- Tree structure for objects: treat individual objects and composites in the same way
- e.g., file directories have entries, each of which may itself be a directory
- e.g., JMenu is a subclass of JMenuItem
- e.g., widgets and containers (Android Views)
- e.g., expression trees

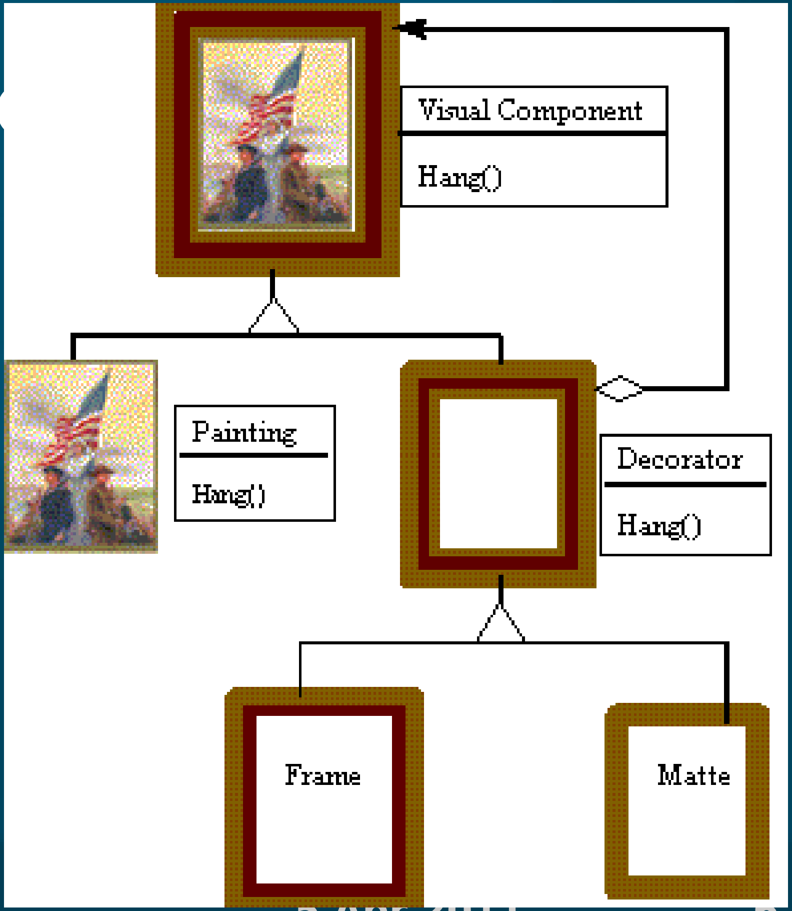




Structural: Decorator

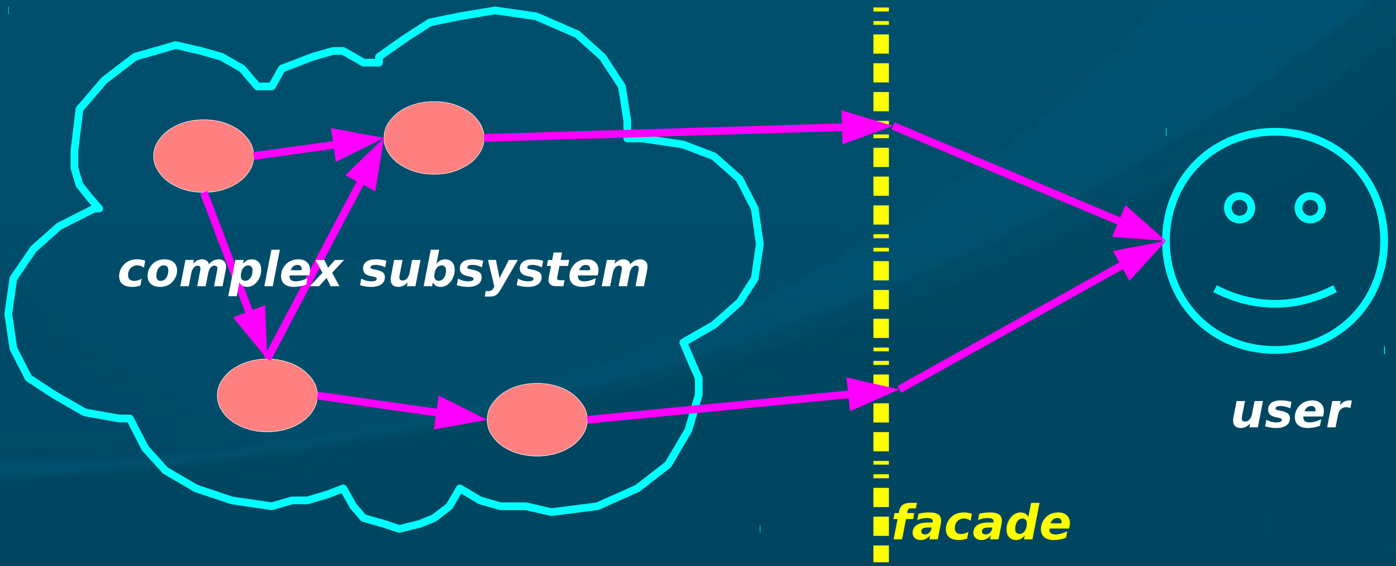
- Dynamically add functionality via a wrapper
  - More flexible than static subclassing
- e.g., JScrollPane for widgets
- e.g., ObjectOutputStream
  a FileOutputStream





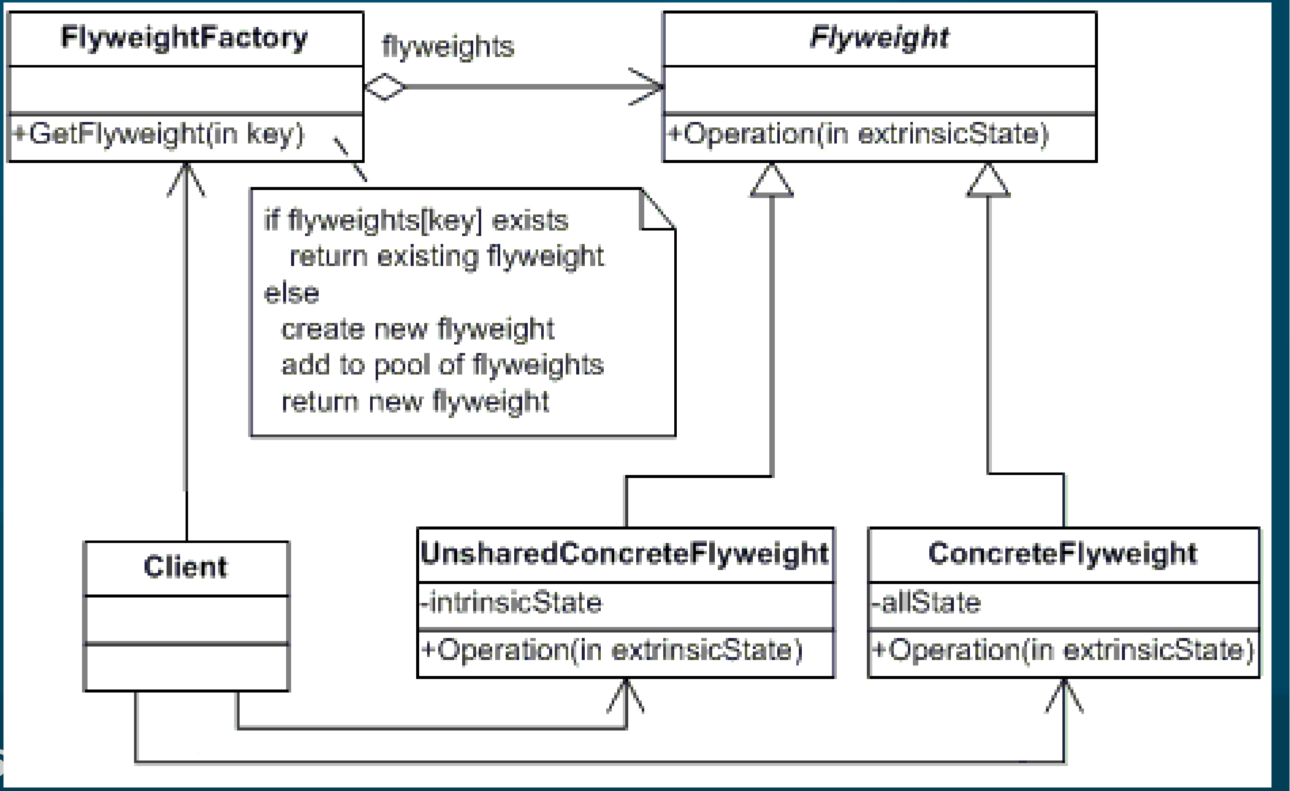
Structural: Facade

- Provide a unified interface to a set of interfaces in a subsystem
  - High-level interface: system is easier to use
  - e.g., web front-end to complex database:
    - want minimal number of widgets, input boxes



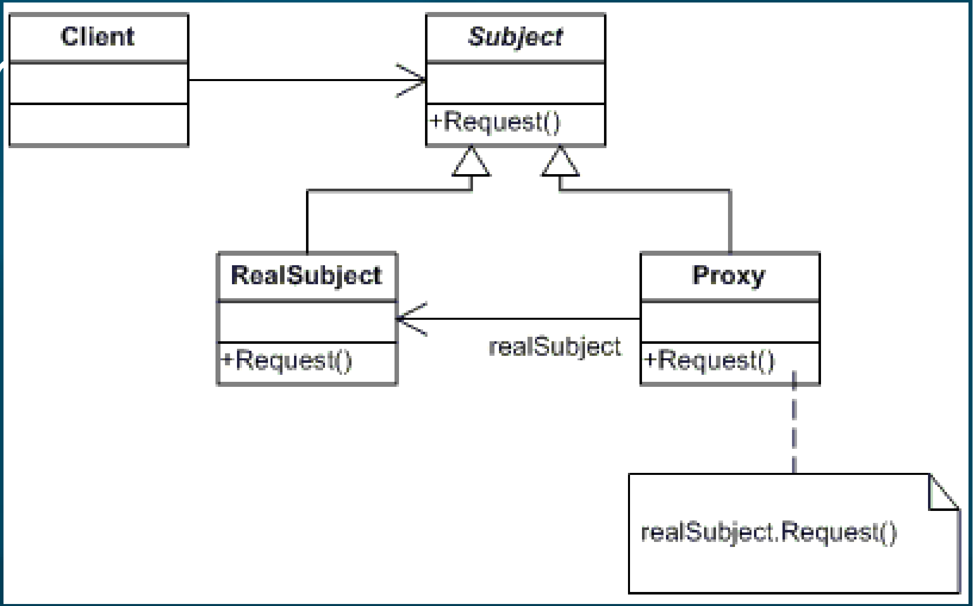
Structural: Flyweight

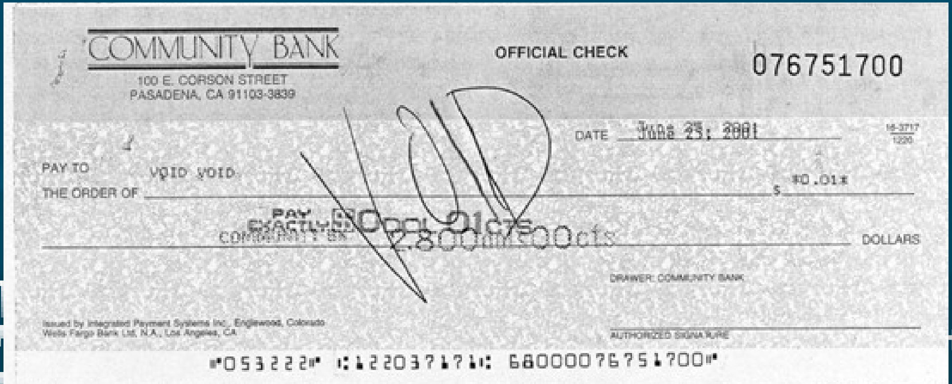
- Draw on-demand from shared pool of light-weight objects
  - May use a factory to create the initial pool
- e.g., thread pool for multithread server
- Array of bank tellers


FlyweightFactory
+GetFlyweight(in key)
flyweights
Flyweight
+Operation(in extrinsicState)
if flyweights[key] exists
return existing flyweight
else
create new flyweight
add to pool of flyweights
return new flyweight
Client
UnsharedConcreteFlyweight
-intrinsicState
+Operation(in extrinsicState)
ConcreteFlyweight
-allState
+Operation(in extrinsicState)

Structural: Proxy

- Surrogate for the real object
- Access to real object is controlled, but clients think they're talking directly to it
- Use superclass over real object and proxy
- Contrast: Adapter, Bridge
- e.g., proxy HTTP server
- e.g., bank cheque




COMMUNITY BANK
100 E. CORSON STREET
PASADENA, CA 91103-3839

OFFICIAL CHECK

076751700

DATE June 23, 2001 16-3717/1220

PAY TO THE ORDER OF VOID VOID $ *0.01*

PAY EXACTLY \$0 DOL 01 CTS DOLLARS

DRAWER: COMMUNITY BANK

Issued by Integrated Payment Systems Inc., Englewood, Colorado
Wells Fargo Bank Ltd, N.A., Los Angeles, CA

AUTHORIZED SIGNATURE

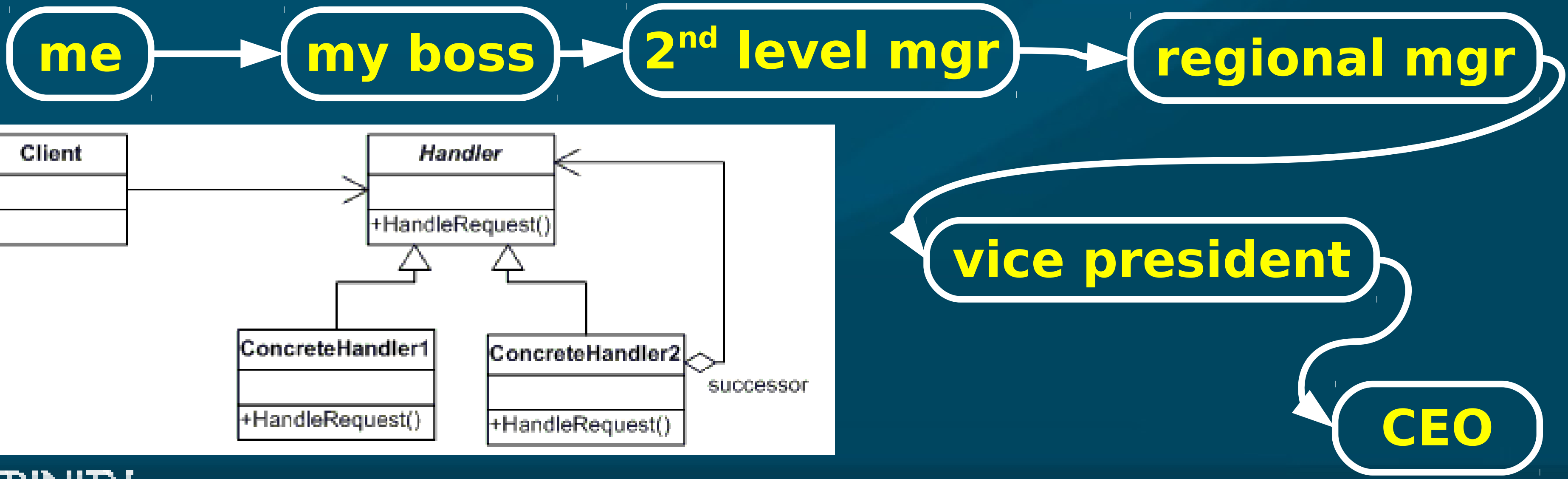
⑈053222⑈ ⑆122037171⑆ 680000767517000⑈

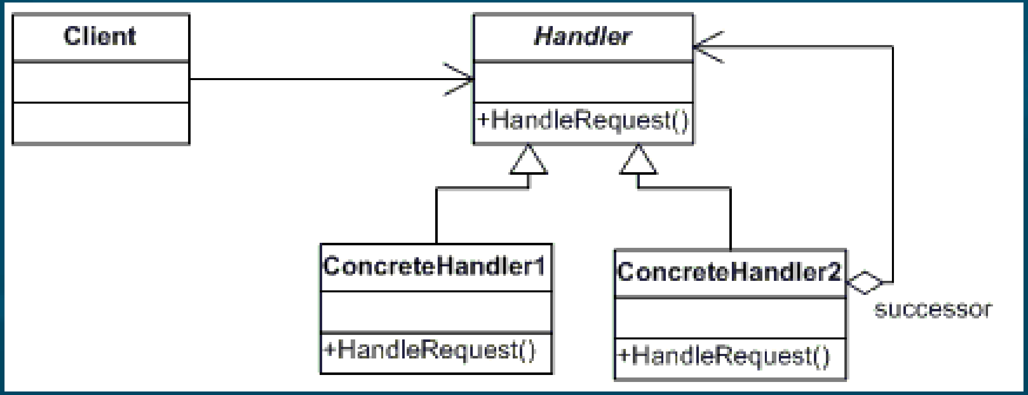
Structural patterns

- Adapter/ wrapper: Convert the interface of a class into another interface clients expect
- Bridge: split abstraction vs. implementation
- Composite: organize objects into trees
- Decorator: dynamically add responsibilities / functionality to an object
- Facade: hide behind simple interface
- Flyweight: use sharing to support large numbers of fine-grained objects efficiently
- Proxy: surrogate/placeholder

Bhv: Chain of responsibility

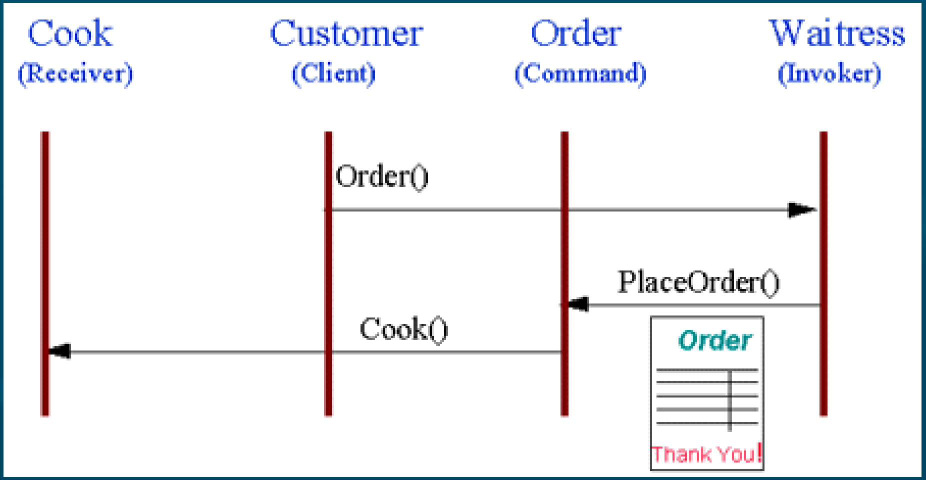
- Decouple sender from receiver by passing request on a chain of intermediate handlers
  - Chain may be reconfigured dynamically
  - Single pipeline, but many possible handlers
- e.g., coin passing through vending machine

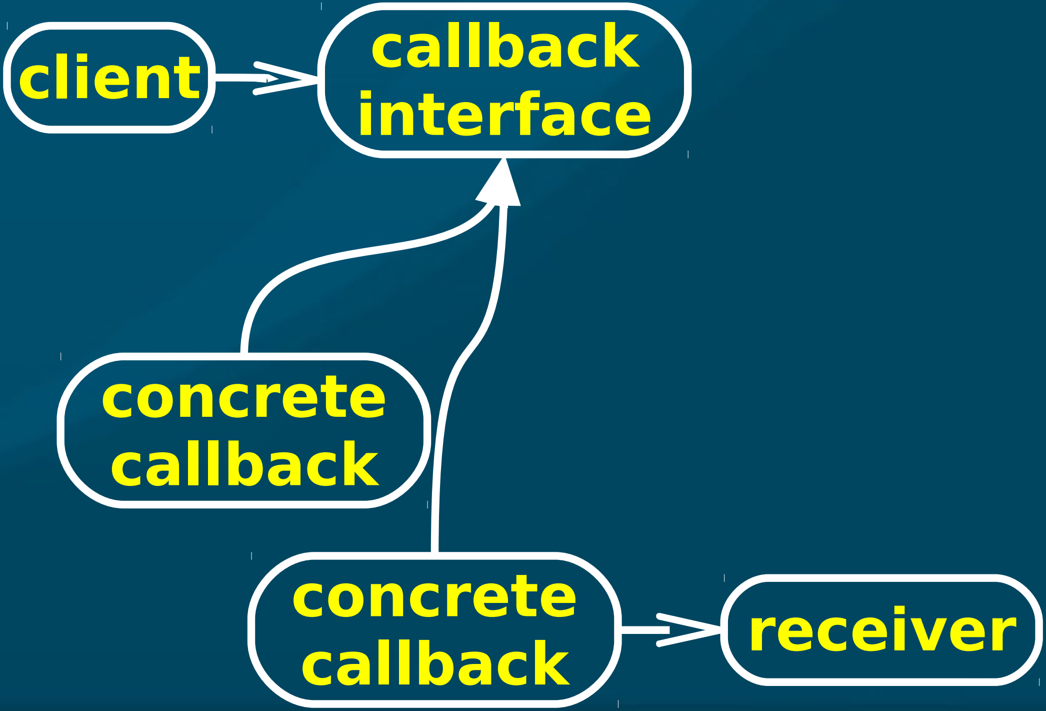




Behavioural: Command

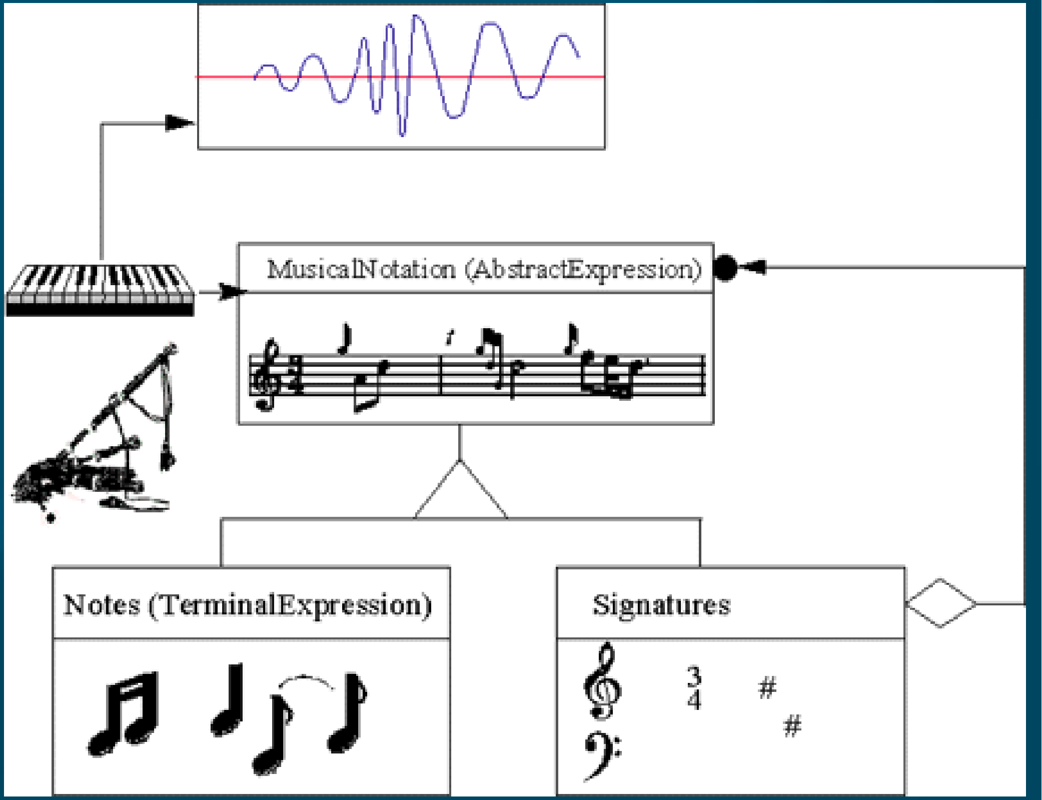
- Encapsulate a request as an object
  - e.g., function objects, callbacks
- Specify: object, method, arguments
- e.g., meal order at restaurant
- Support undo/redo





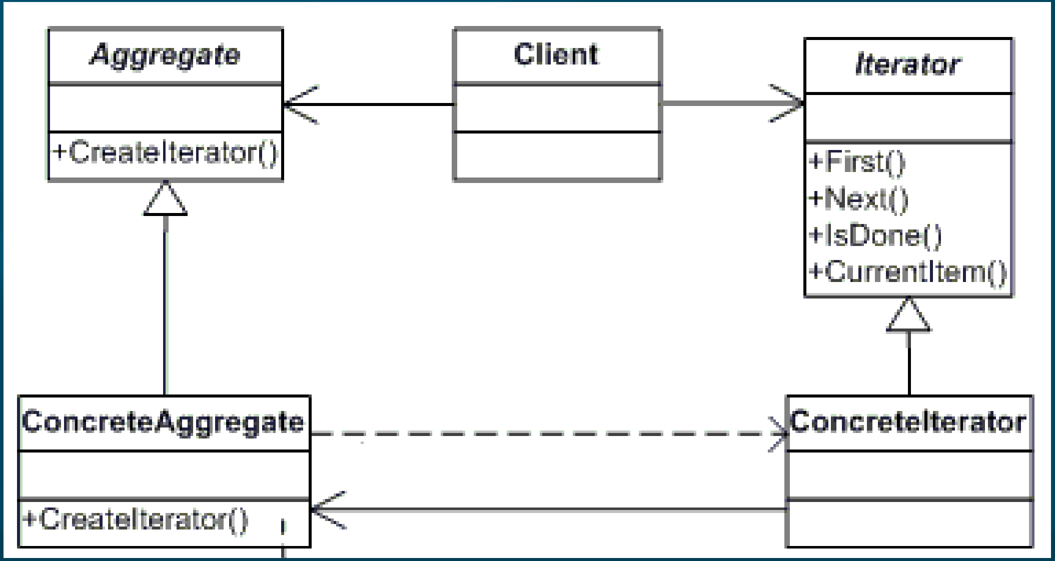
Behavioural: Interpreter

- Given a domain-specific language, define a grammar for the language and an engine to translate into objects
- Vocabulary + syntax
- e.g., parse config file
- e.g., read music → produce sound
- Useful for repeated, similar problems in a well-defined domain


MusicalNotation (AbstractExpression)
Notes (TerminalExpression)
Signatures

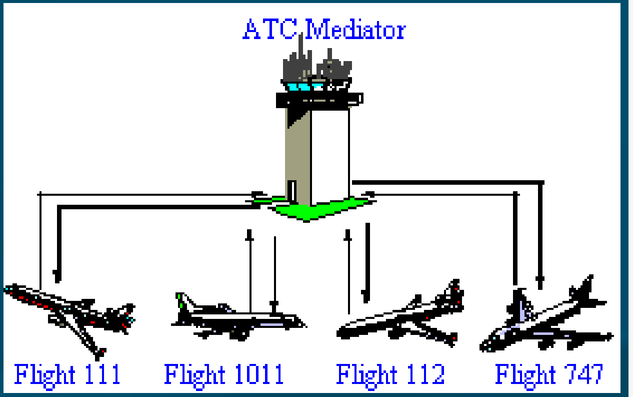
Behavioural: Iterator

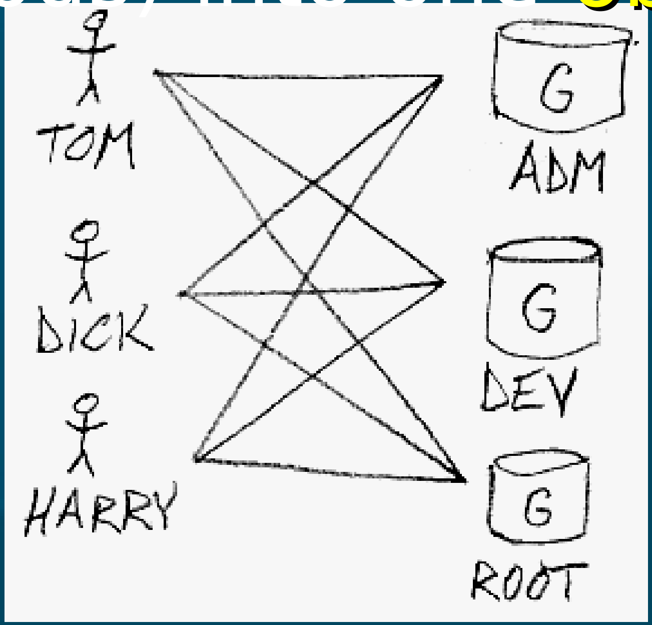
- Abstract interface to traverse a collection
- Hide how the collection is stored
- Client interface: first, next, isDone
- e.g., secretary knows her own filing system; boss only needs ask for “next document”
- e.g., for/each loop through dictionaries
  - Order irrelevant



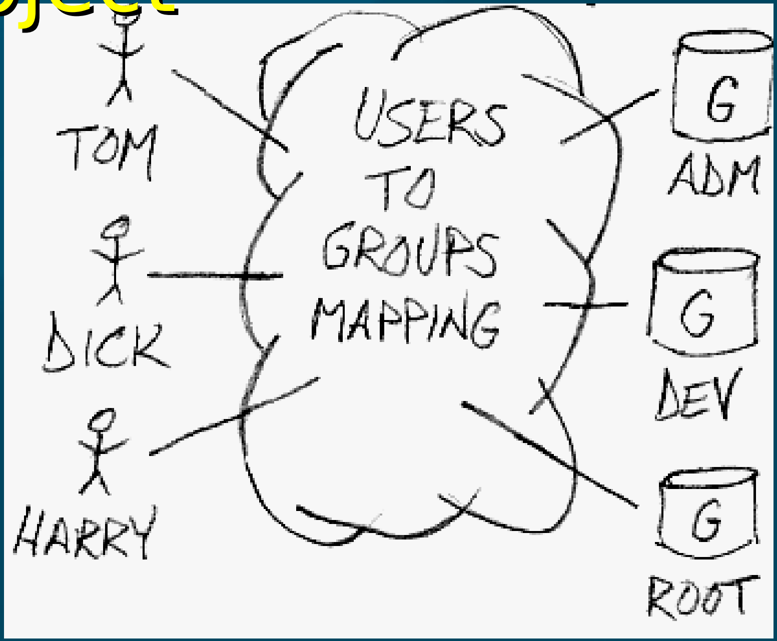
Behavioural: Mediator

- Simplify many-to-many relationships: one central object that all actors interact with
  - Loose coupling of peers
- Encapsulate many interactions (e.g., methods) into one object
- e.g., ATC





w/o mediator



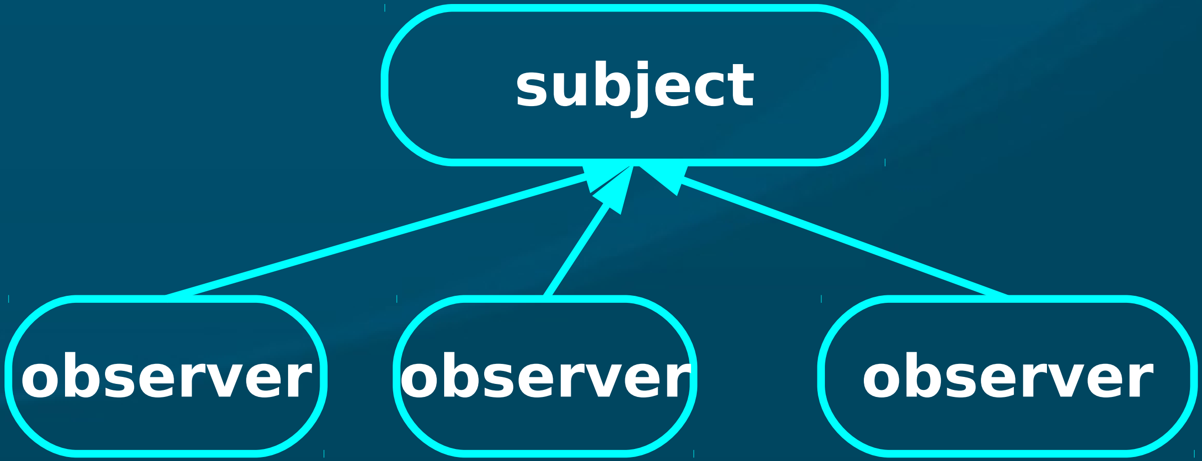
with mediator

Behavioural: Memento

- Transparently save/restore object state
  - e.g., pickling/serialization
- Allows undo/redo, checkpoint/snapshot, etc.
- Originator: object that can snapshot
- Caretaker: requests snapshot fr Originator, keeps Memento, later restores Originator
- Memento: object representing Originator state

Behavioural: Observer

- One-to-many dependency among objects: When the subject changes state, all its observers are notified and updated
  - e.g., TV/radio broadcast
  - e.g., server message “send to all” clients
  - e.g., RSS feeds



Behavioural patterns

- Chain of responsibility: uncouple sender from receiver via chain of intermediaries
- Command: make requests into objects
- Interpreter: define macro language + parser
- Iterator: access all elements of a collection
- Mediator: encapsulates the interactions of a set of objects → loose coupling
- Memento: save/restore state of object
- Observer: viewers decoupled from subject