Design Defects and Restructuring

Lecture 10

Sat, Nov 27, 2021

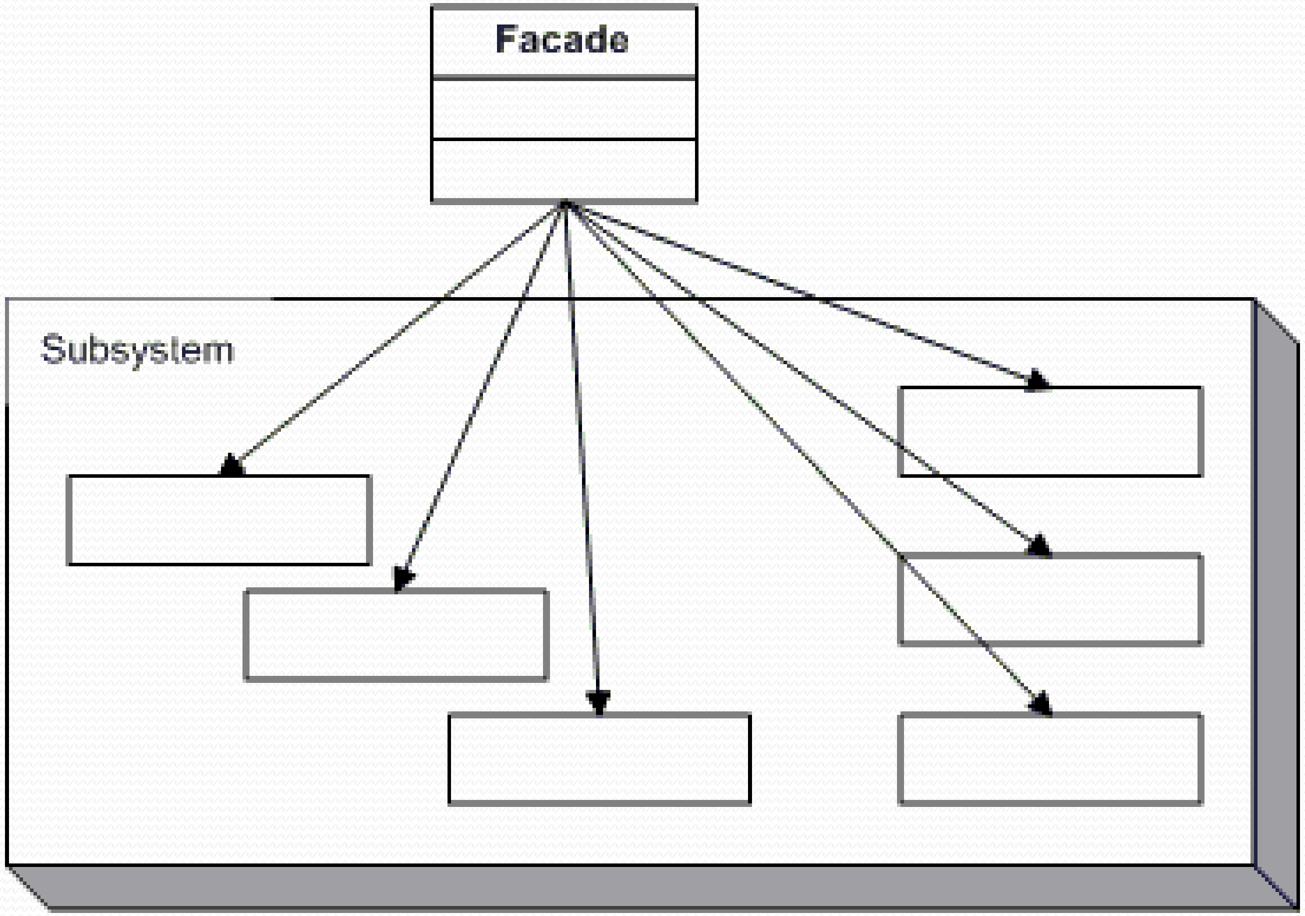
Structural Patterns

- Adapter
- Bridge
- Composite
- Decorator
- Façade
- Flyweight
- Proxy

Façade

- Intent
  - Provide a unified interface to a set of interfaces in a subsystem
  - Façade defines a higher-level interface that makes the subsystem easier to use
- Applicability
  - You want to provide a simple interface to a complex subsystem
  - There are many dependencies between clients and the implementation classes of an abstraction
    - Introduce a facade to decouple the subsystem from clients and other subsystems, thereby promoting subsystem independence and portability
  - You want to layer your subsystems

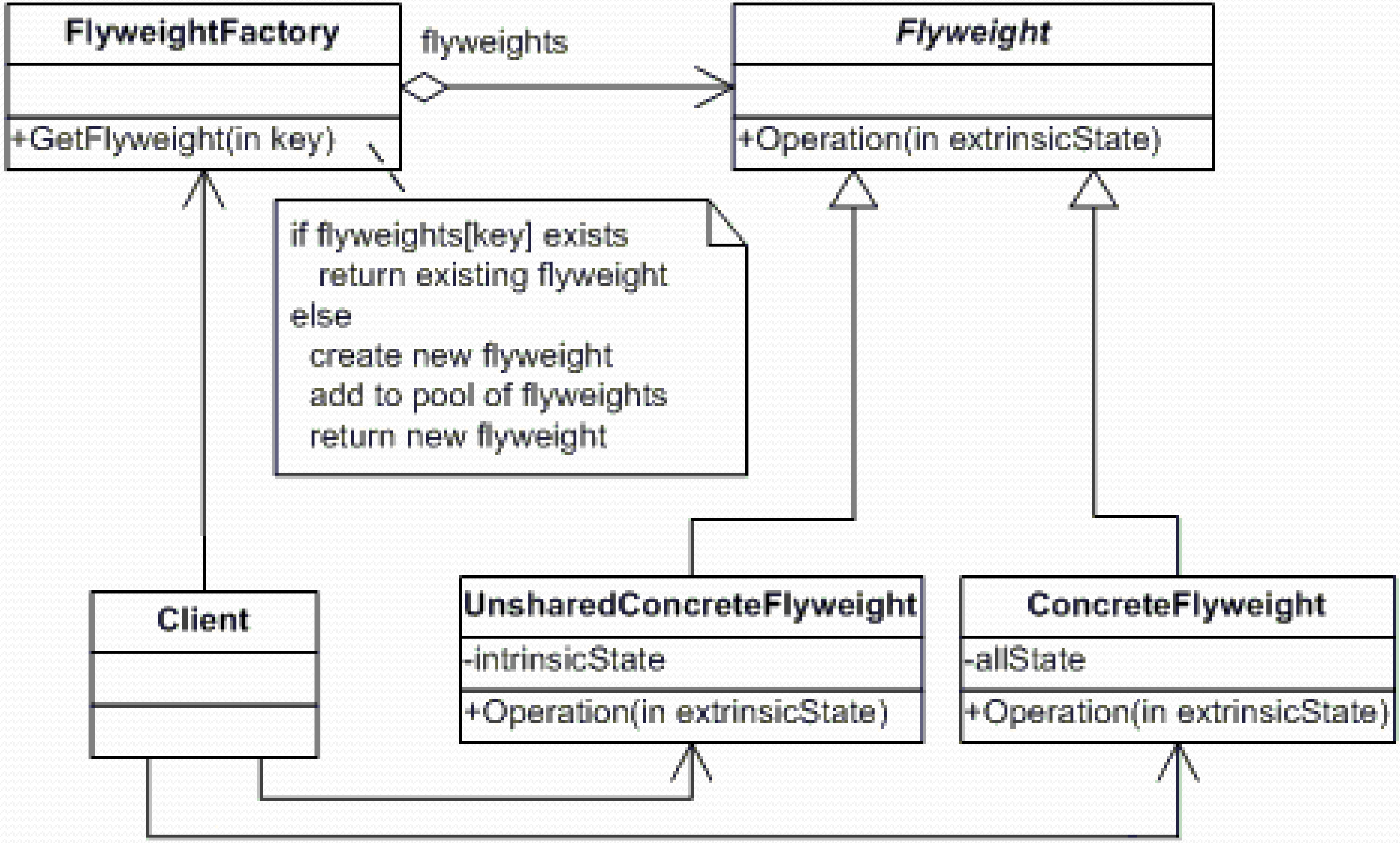
Façade



Flyweight

- Intent
  - Use sharing to support large numbers of fine-grained objects efficiently
- Applicability
  - An application uses a large number of objects
  - Storage costs are high because of the sheer quantity of objects
  - Most object state can be made extrinsic
  - Many groups of objects may be replaced by relatively few shared objects once extrinsic state is removed
  - The application does not depend on object identity
    - Since flyweight objects may be shared, identity tests will return true for conceptually distinct objects

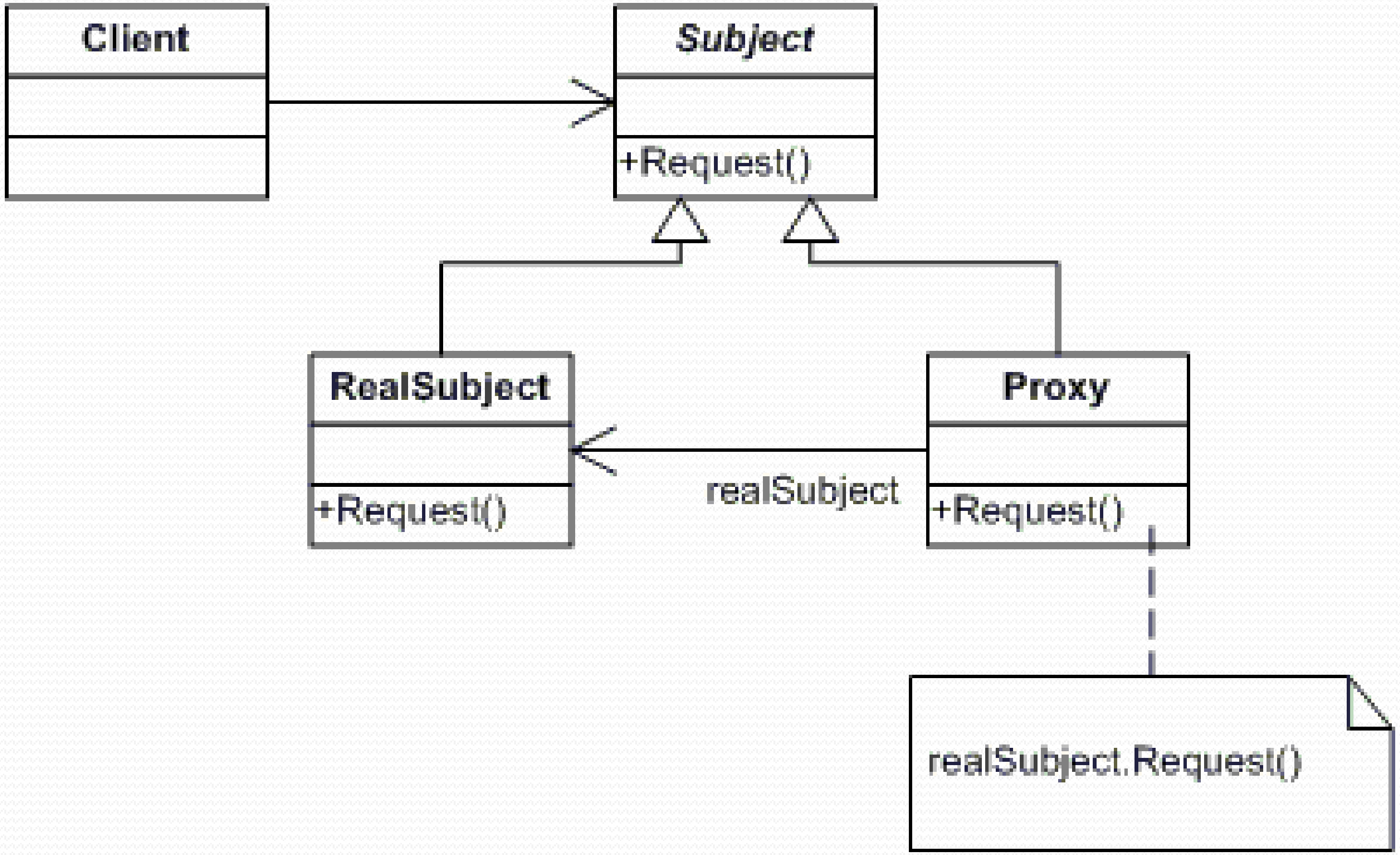
Flyweight


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)

Proxy

- Intent
  - Provide a surrogate or placeholder for another object to control access to it
- Applicability
  - A remote proxy provides a local representative for an object in a different address space
  - A virtual proxy creates expensive objects on demand
  - A protection proxy controls access to the original object
  - A smart reference is a replacement for a bare pointer that performs additional actions when an object is accessed

Proxy


Client
Subject
+Request()
RealSubject
+Request()
realSubject
Proxy
+Request()
realSubject.Request()

Behavioral Patterns

- Chain of Responsibility
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method
- Visitor

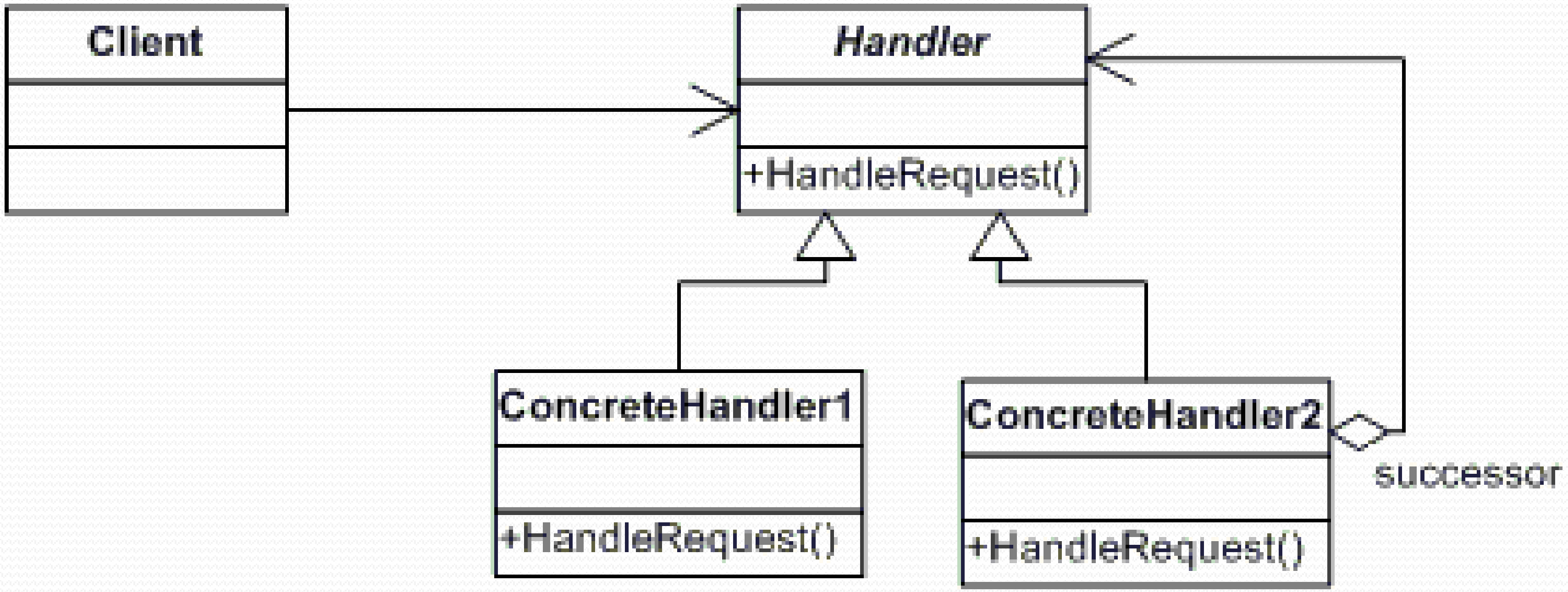
Chain of Responsibility

- Intent
  - Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request
  - Chain the receiving objects and pass the request along the chain until an object handles it

Chain of Responsibility

- Applicability
  - More than one object may handle a request, and the handler isn’t known a priori
    - The handler should be ascertained automatically
  - You want to issue a request to one of several objects without specifying the receiver explicitly
  - The set of objects that can handle a request should be specified dynamically

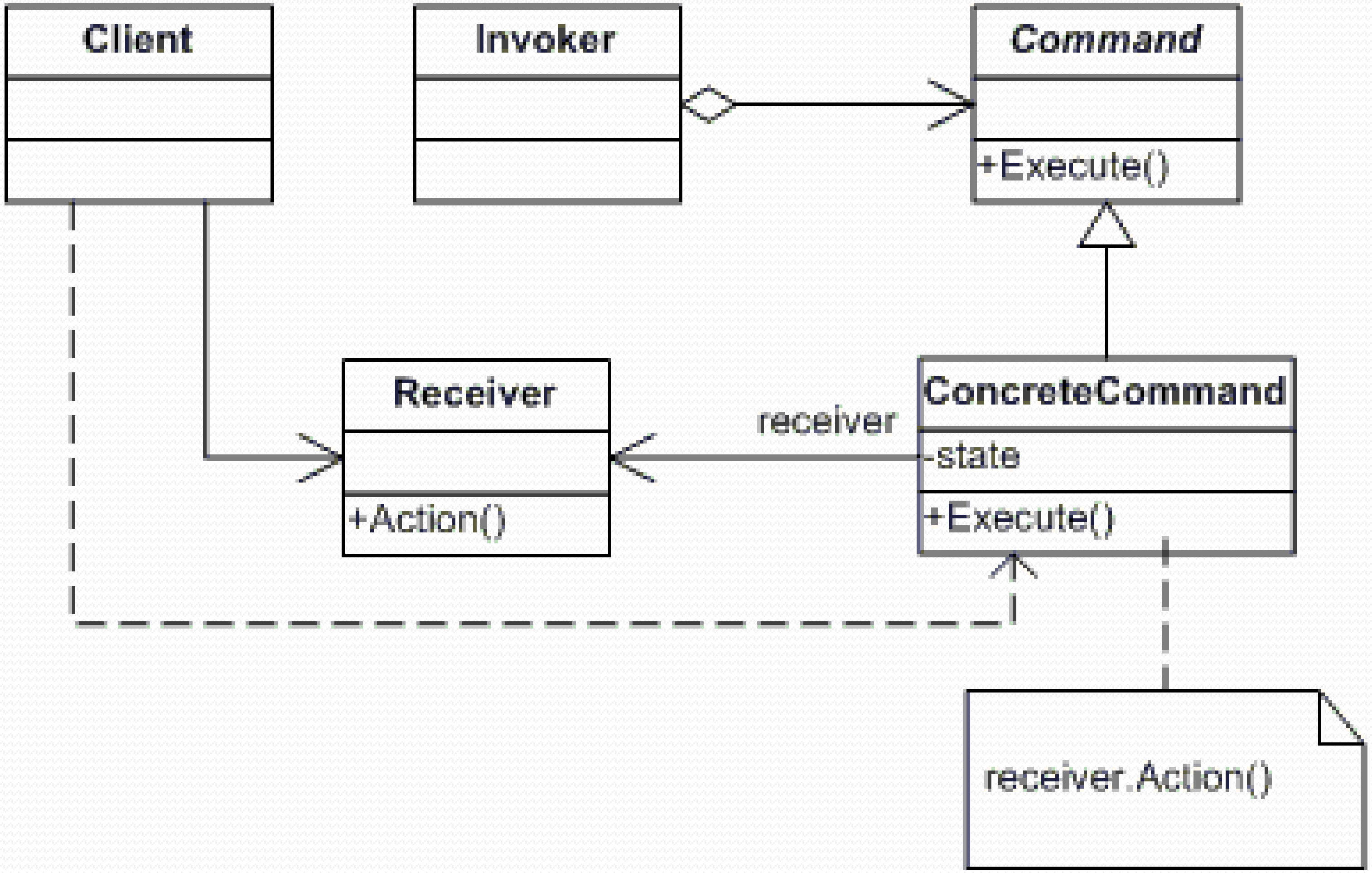
Chain of Responsibility



Command

- Intent
  - Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations
- Applicability
  - Parameterize objects by an action to perform
    - Commands are an object-oriented replacement for callbacks
  - Specify, queue, and execute requests at different times
  - Support undo
  - Support logging changes so that they can be reapplied in case of a system crash
  - Structure a system around high-level operations built on primitives operations

Command


Client
Invoker
Command
+Execute()
Receiver
+Action()
receiver
ConcreteCommand
-state
+Execute()
receiver.Action()