

CSSE2003

Software Engineering Studio

Semester 2, 2009

17: Software Patterns - 3

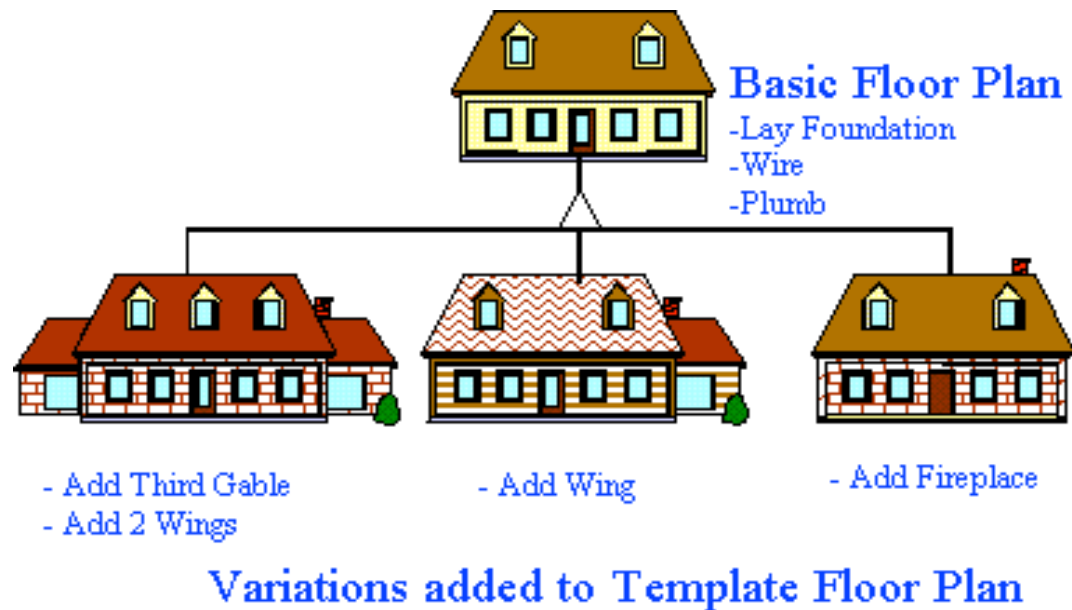
Lecture Summary

- ◆ Behavioural patterns
 - Operation Patterns
 - ◆ Template
 - ◆ Command
 - Extension patterns
 - ◆ Iterator
 - ◆ Visitor
 - Responsibility patterns
 - ◆ Observer
 - ◆ Mediator

Template Pattern

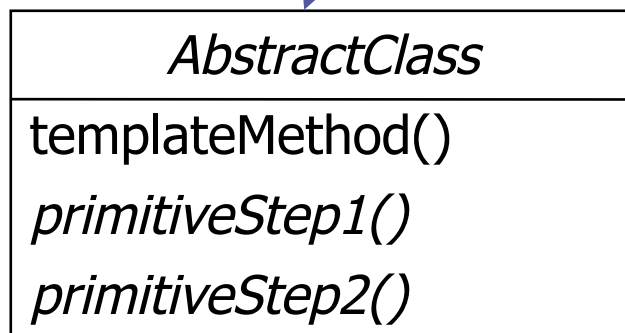
- ◆ Defines the skeleton of an algorithm in an operation, deferring some steps to methods defined in subclasses.
- ◆ The subclasses define the detail of these steps without changing the algorithm's structure.

- ◆ SW example:
any class with abstract methods is a form of the Template pattern.

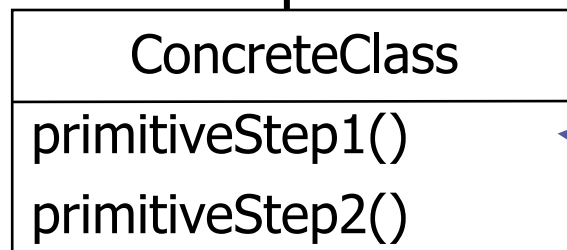


Template Pattern Structure

defines abstract primitive operations that concrete subclasses define and implements a template method that invokes the primitive operations



primitiveStep1();
primitiveStep2()

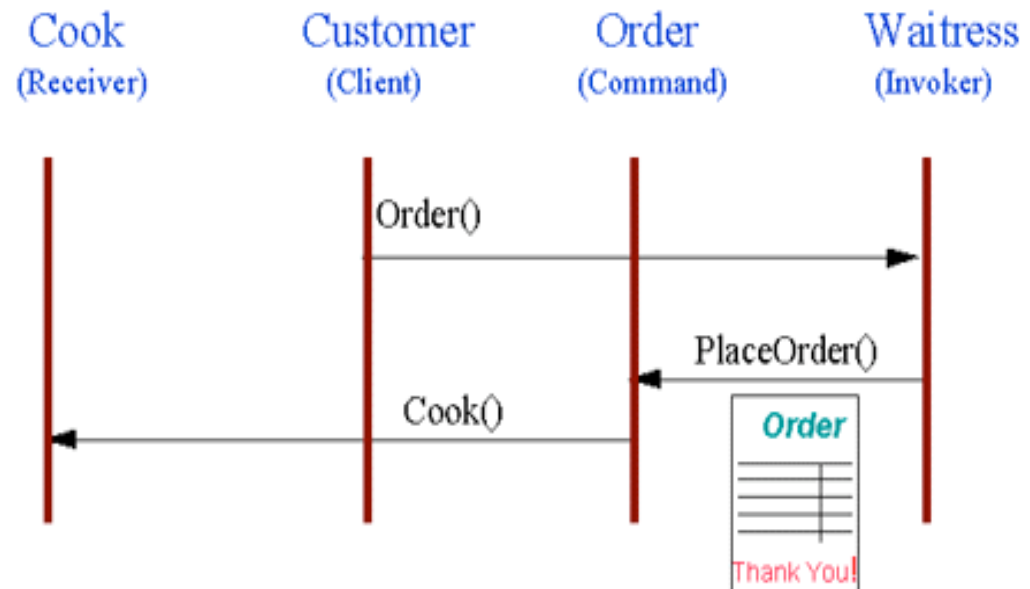


implements the primitive operations to perform subclass-specific steps of the algorithm.

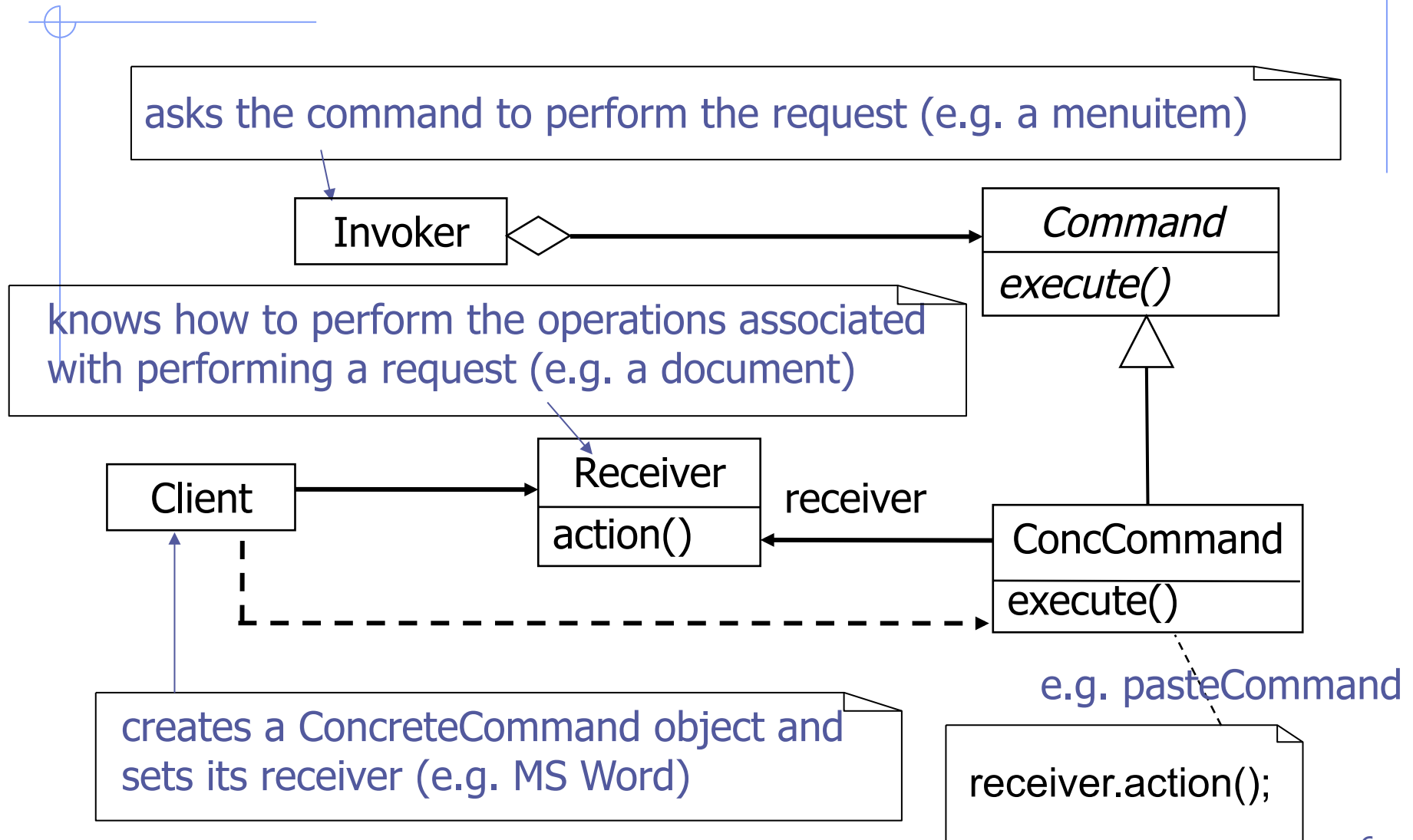
Command Pattern

- ◆ Encapsulates a request as an object and gives it a **known public** interface.
- ◆ Commonly used to keep the application and user interface objects completely separate from the actions that they initiate, e.g. menu entries and buttons.

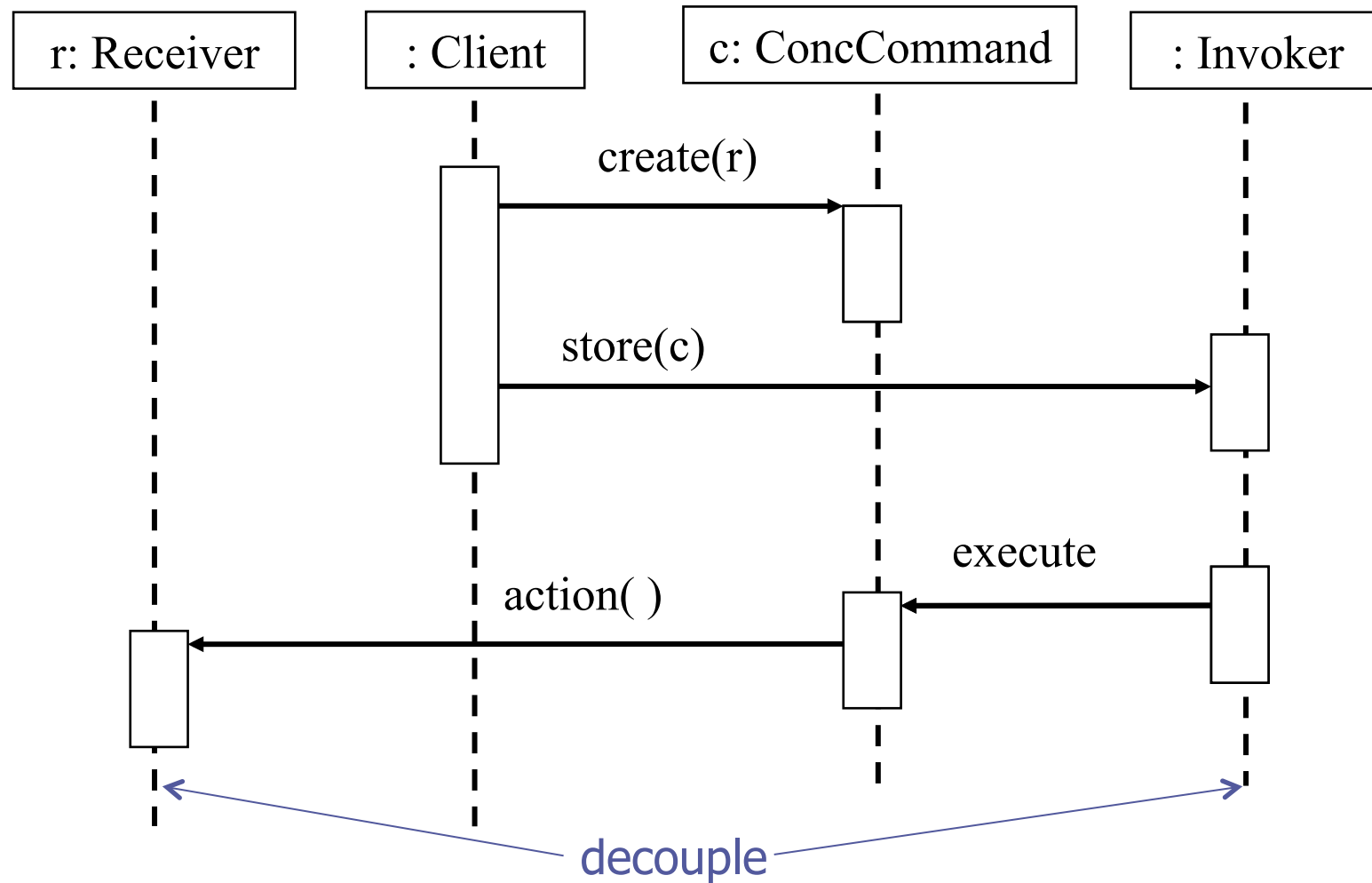
- ◆ Command objects are also convenient for supporting “undo” capabilities or transactions.



Command Pattern Structure



Command Pattern Interactions



Extension Mechanisms

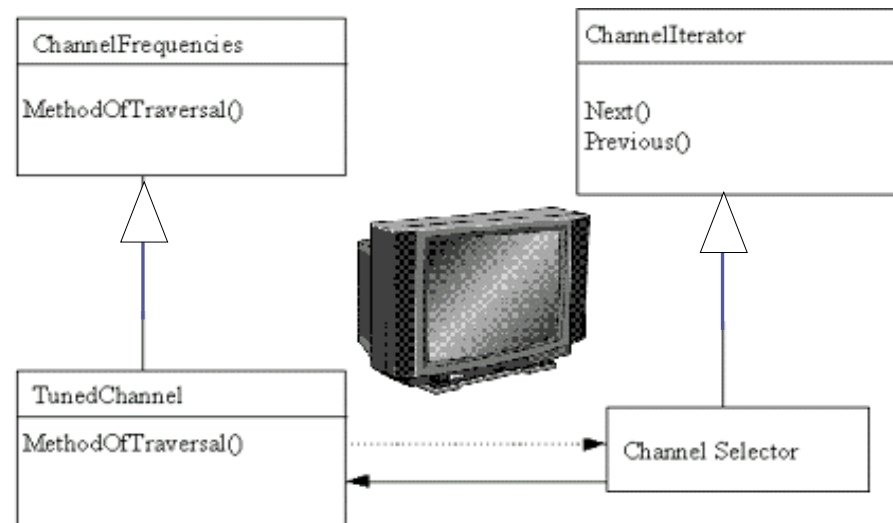
- ◆ Extension is adding to an existing system, typically a class, an interface or a method.
- ◆ Extension can be achieved by:
 - inheritance of an existing class - an instance of the new subclass should function as an instance of the superclass (Liskov Substitution principle)
 - delegation to reuse behaviour - methods forward calls to identical operations supplied by another object

Extension-oriented Patterns

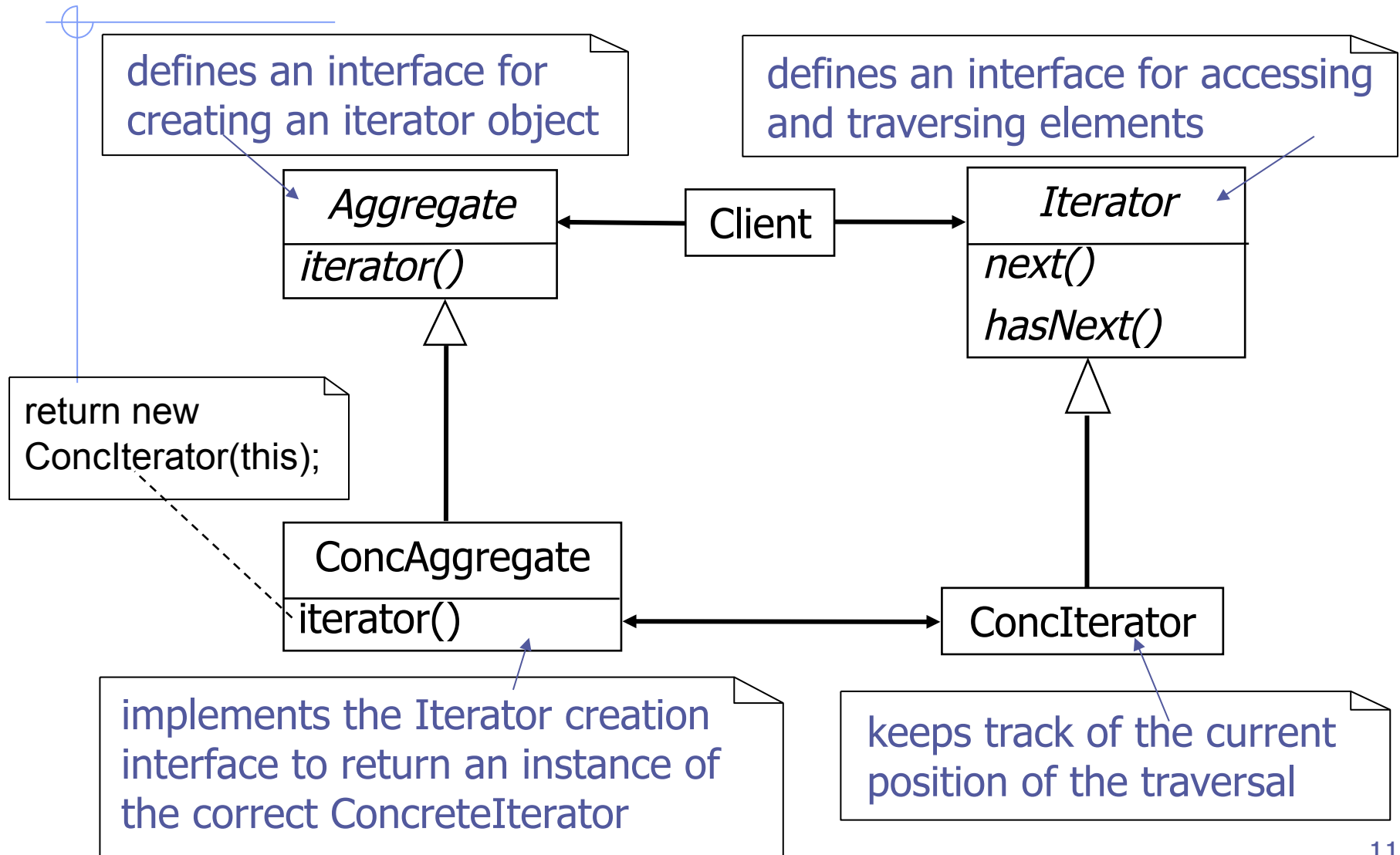
- ◆ **Iterator** - provide a way to access a collection of a class that you create.
- ◆ **Visitor** - allow for the addition of new operations to a class without changing the class.

Iterator Pattern

- ◆ Provides a way to access the elements of an aggregate sequentially without exposing its internal representation.
- ◆ The key idea is to separate responsibility for traversal from the aggregate thus simplifying the aggregate.
- ◆ SW example: Java provides the Iterator interface as its preferred iterator.

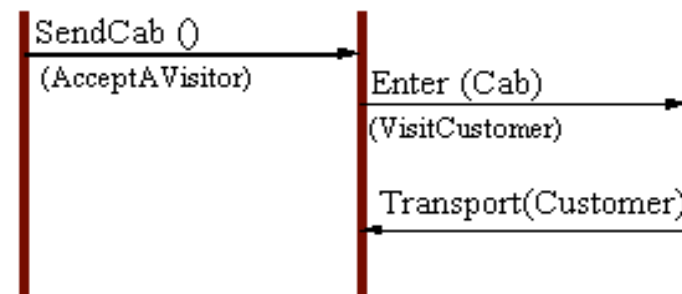


Iterator Pattern Structure

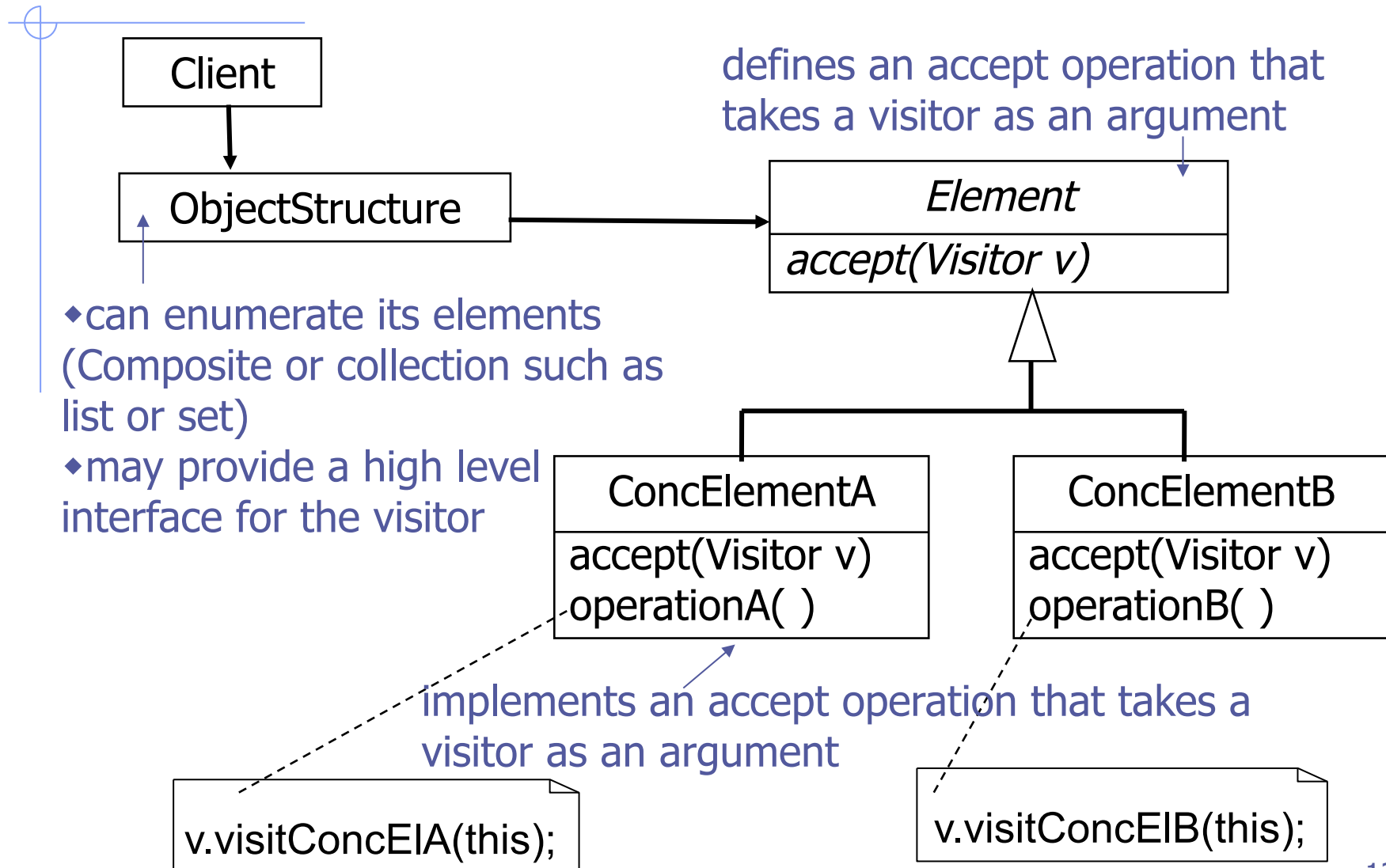


Visitor Pattern

- ◆ Separates an operation to be performed on the elements of an object structure.
- ◆ Lets you define a new operation without changing the classes of the elements.
- ◆ Use this pattern when there are many distinct and unrelated operations to be performed on an object structure.
- ◆ This avoids changing the interface of the object structure elements when a new operation is defined.

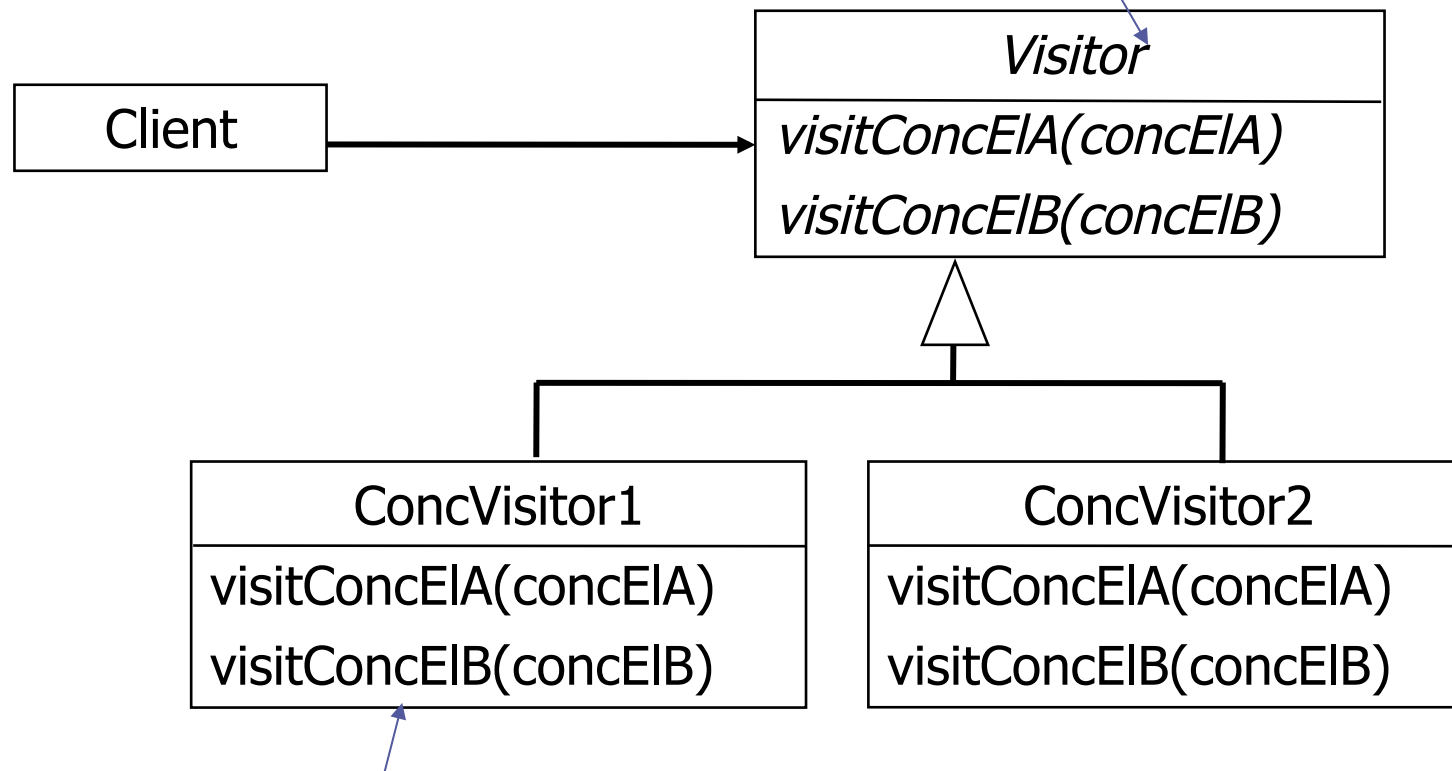


Visitor Structure - 1



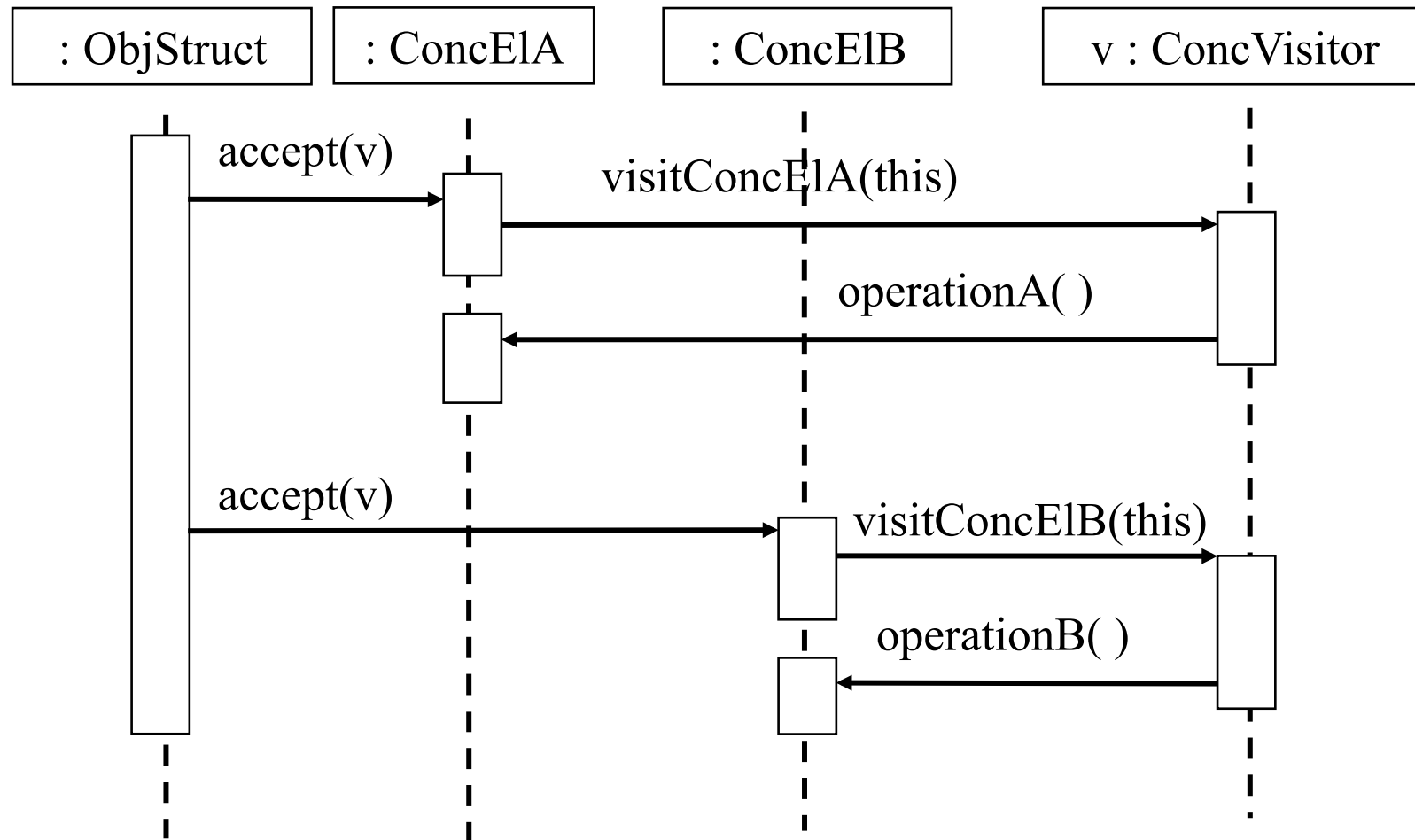
Visitor Structure - 2

declares a visit operation for each class of ConcreteElement in the object structure



- implements each operation declared by the Visitor
- its state often accumulates the results of the traversal

Visitor Pattern Interactions



Responsibility Patterns

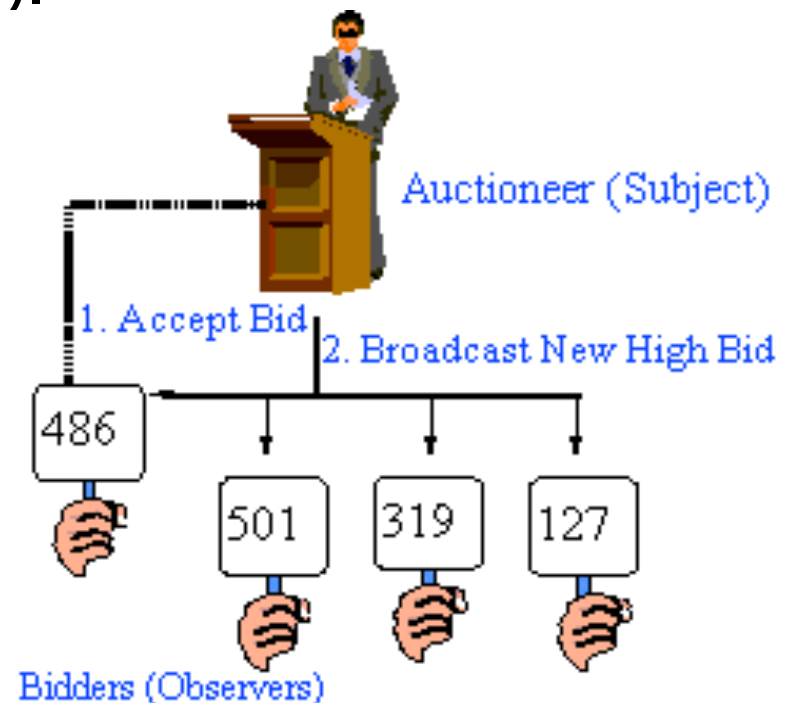
- ◆ This set of responsibility patterns counterbalance the normal distributed responsibility by centralising some responsibilities.
- ◆ Every design pattern solves a problem in a context; responsibility-oriented patterns address contexts where you need to deviate from the standard approach of distributing responsibility.

Observer - decouple an object from knowledge of what other objects depend on it.

Mediator - centralise the interactions between a set of objects.

Observer Pattern

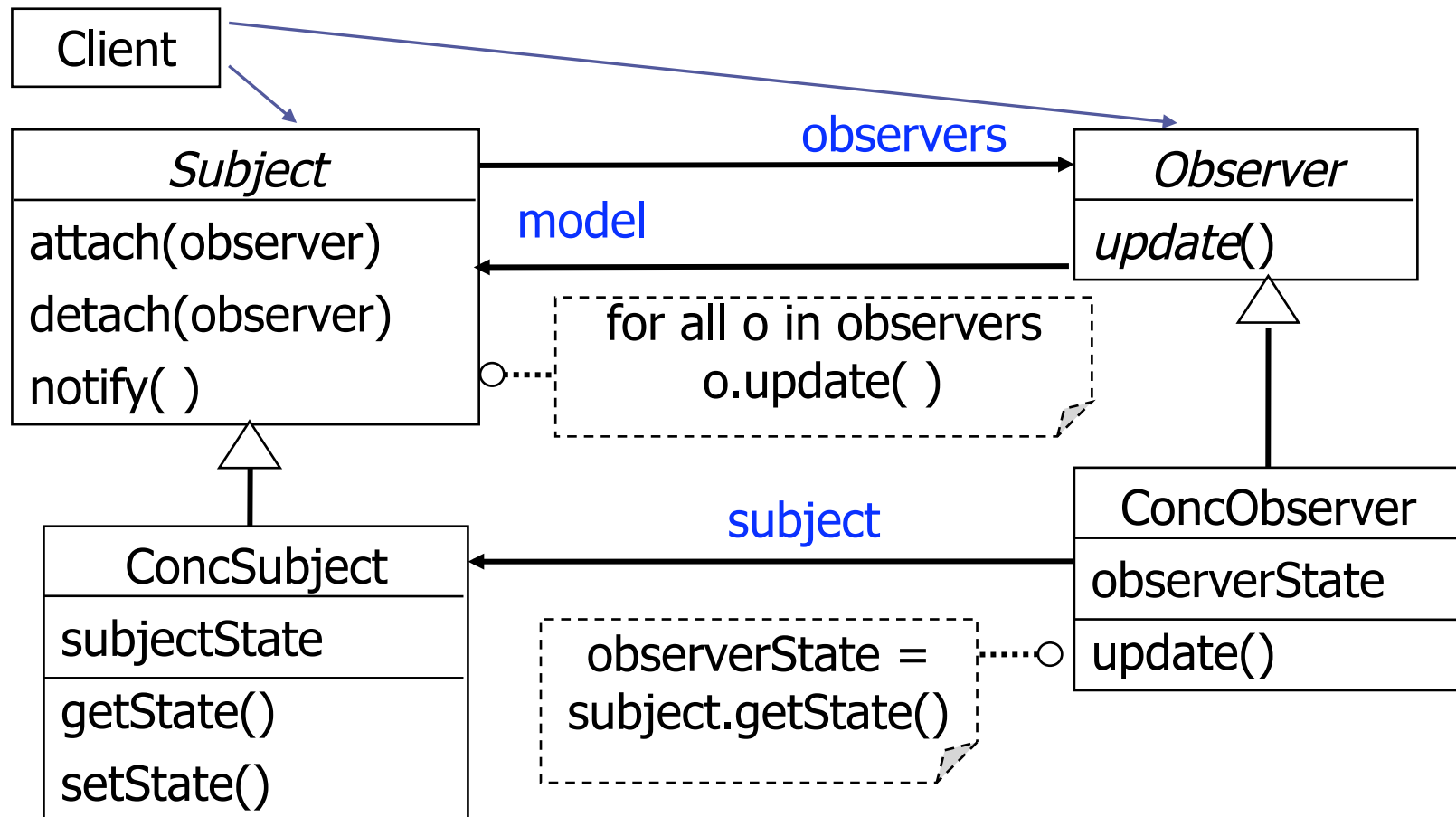
- ◆ Defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically (known as publish-subscribe).
- ◆ The subject object doesn't know the details of any of its observers.
- ◆ SW example: MVC structure where user interface elements are detached from the underlying application data.



Observer Participants

- ◆ Subject – knows its observers (any number) and provides an interface for them to attach and detach.
- ◆ Observer – defines an updating interface for objects to be notified of changes in a subject.
- ◆ ConcreteSubject – stores state of interest to ConcreteObservers.
- ◆ ConcreteObserver – maintains a reference to a ConcreteSubject and implements the Observer interface.

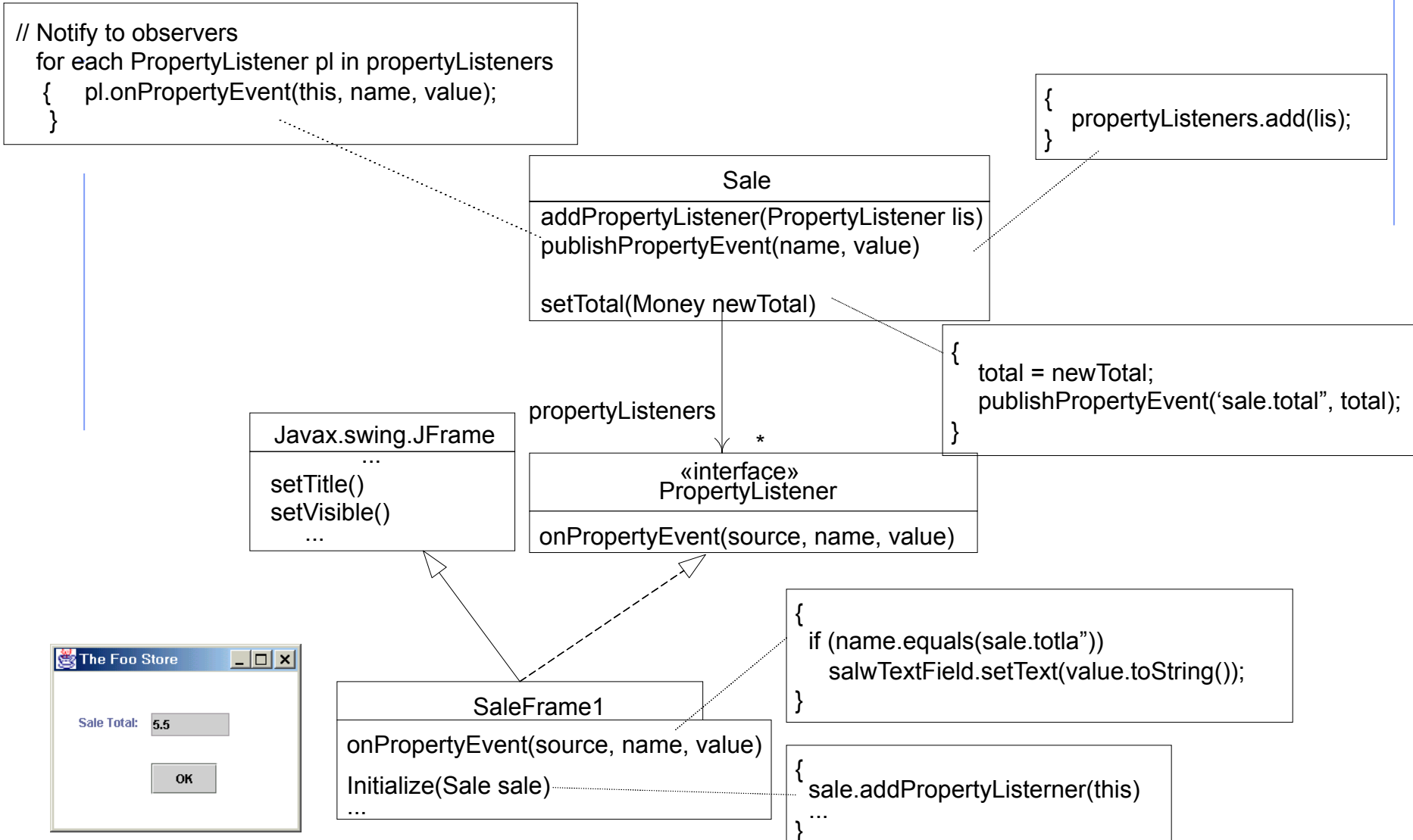
Observer Pattern Structure



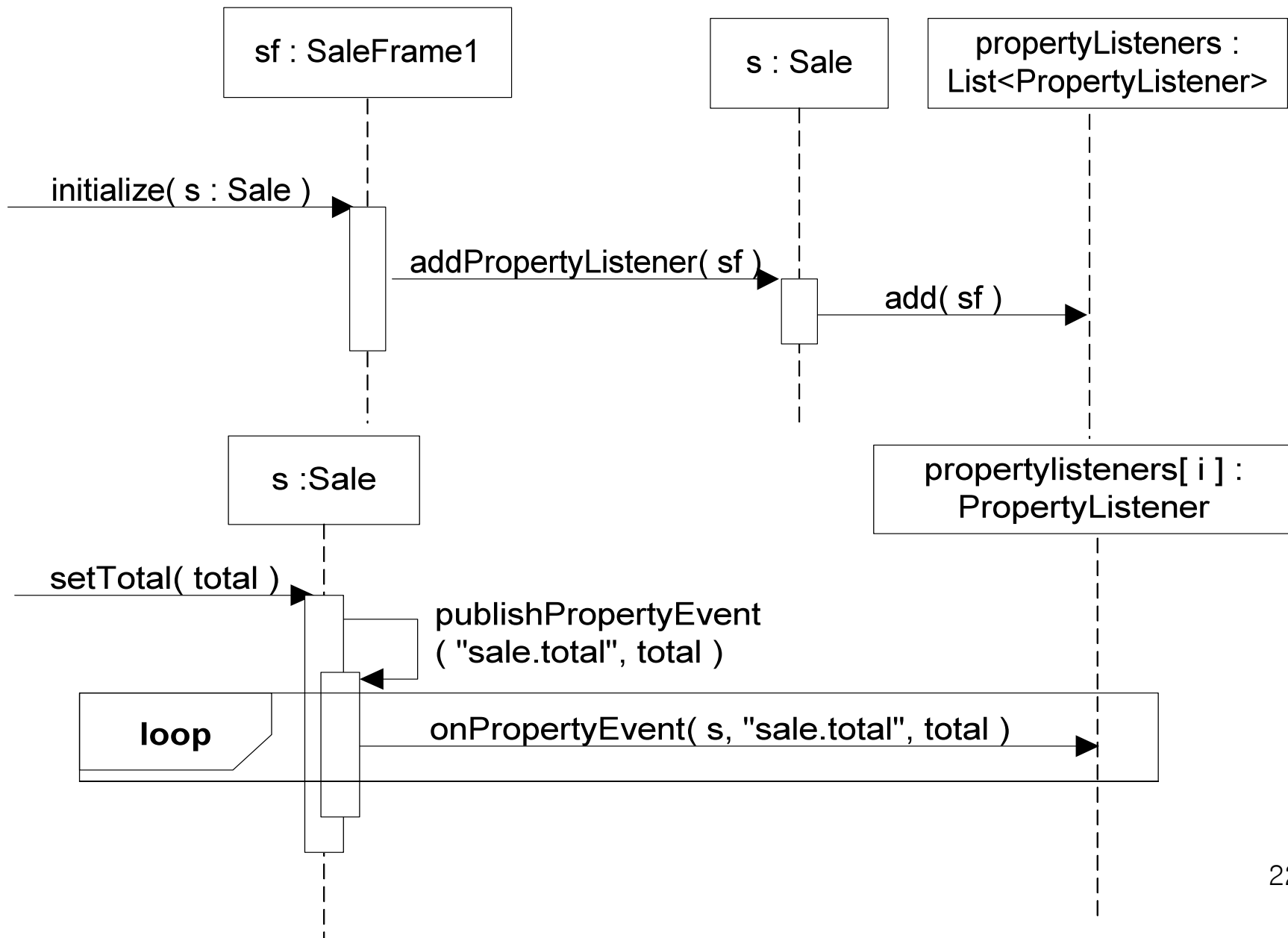
Observer Consequences

- ◆ Java provides the Observable class and the Observer interface in the java.util library.
- ◆ It is not always possible in Java for a concrete subject class to be a subclass of Observable.
- ◆ In that situation, the concrete subject class can contain an Observable object (delegation).

Observer Example

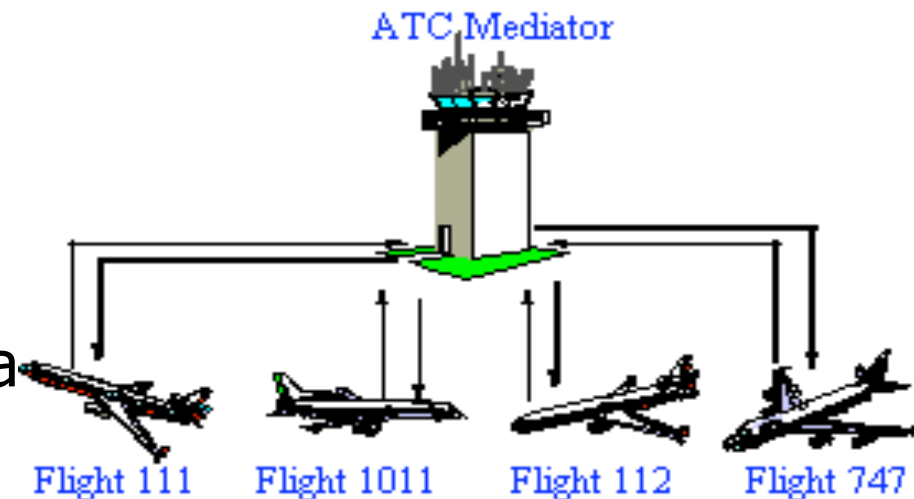


Interaction Model Using Observer

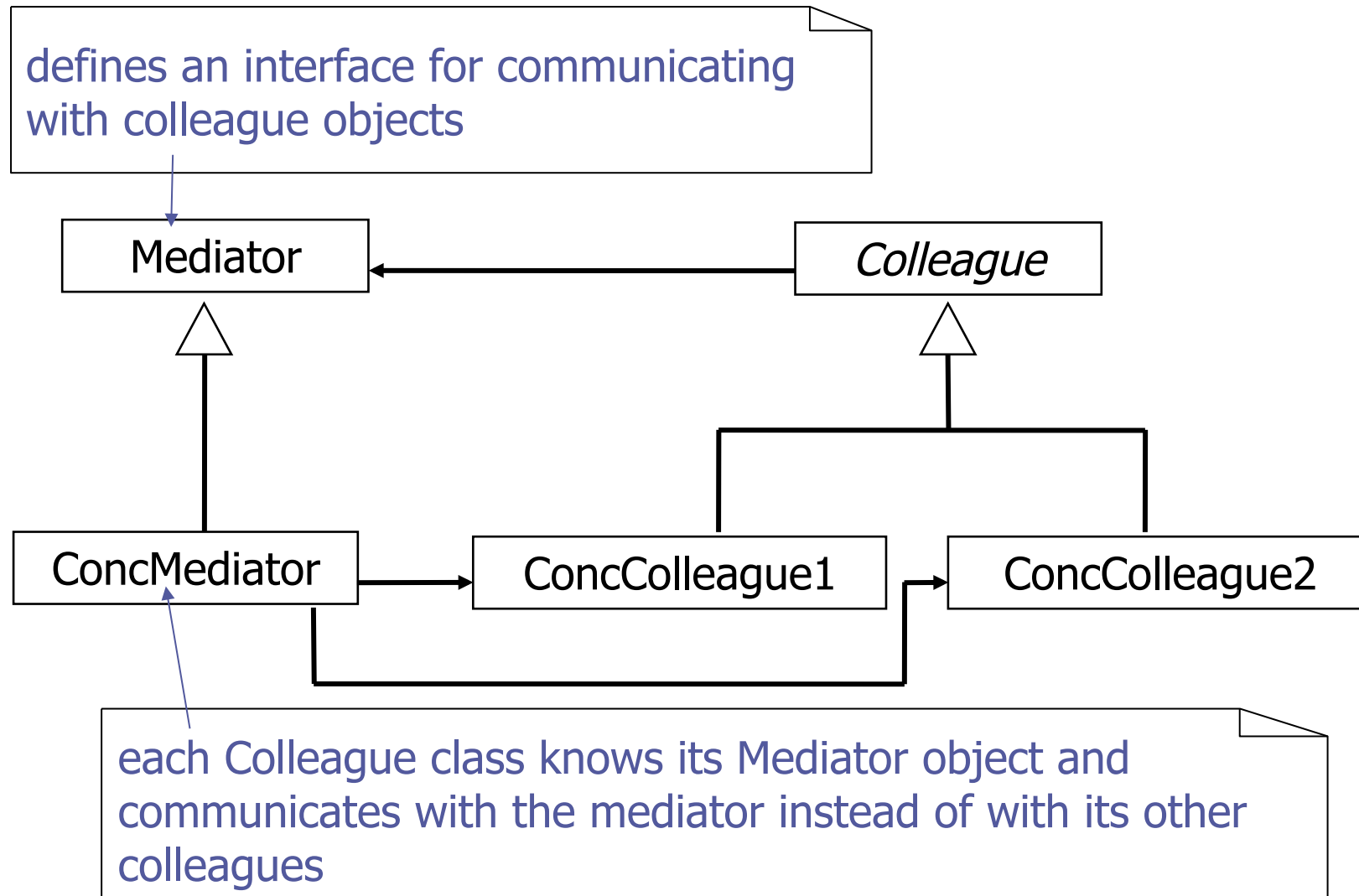


Mediator Pattern

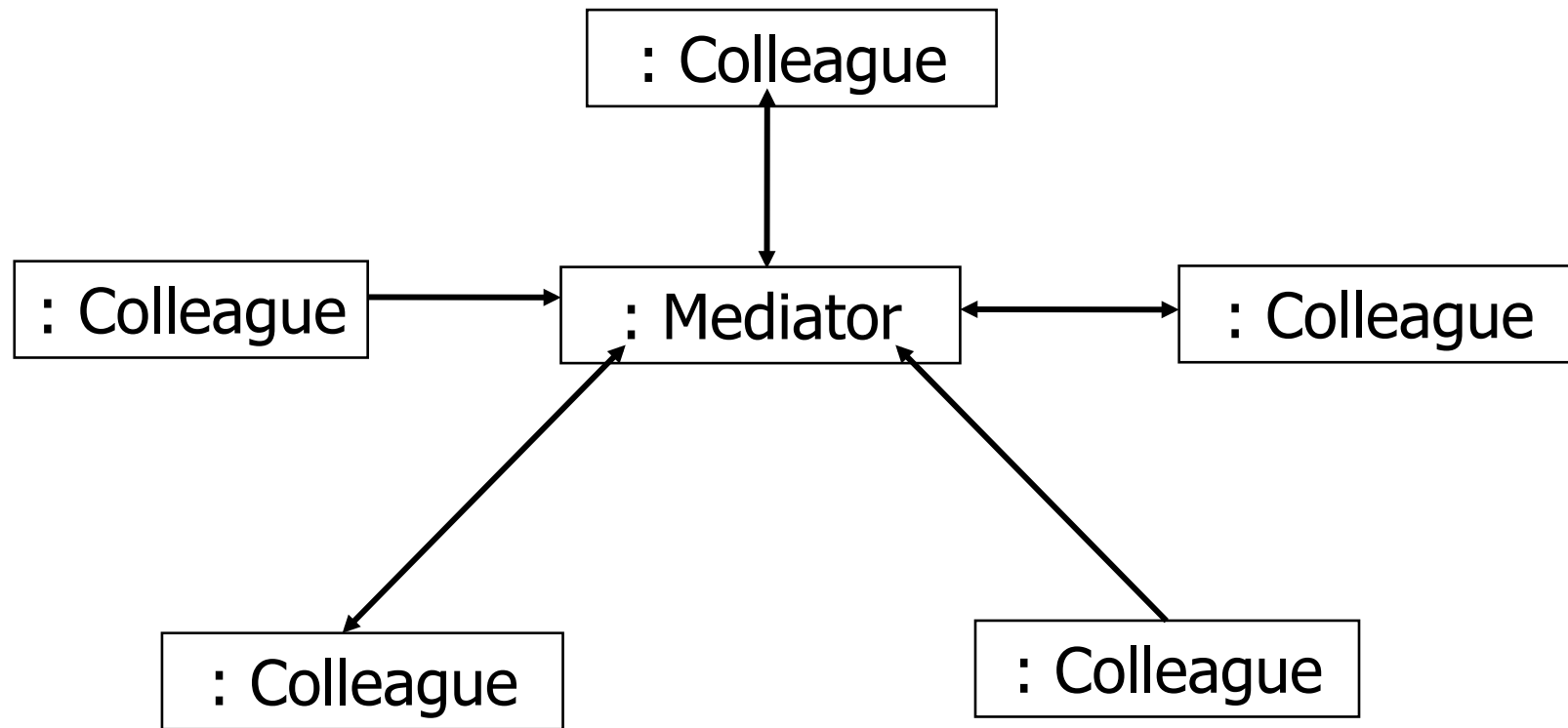
- ◆ Replaces many-to-many interactions with **one-to-many** interactions between the mediator and its colleagues. The one-to-many relationships are easier to understand, maintain and extend.
- ◆ Promotes **loose coupling** by minimising explicit referencing between objects that need to interact.
- ◆ SW example: user interface where there are dependencies between widgets, e.g. a button is disabled if a text field is empty.



Mediator Pattern Structure



Mediator Example



Mediator Consequences

- ◆ The individual components should be simpler and more generic but the Mediator class is often application specific and difficult to reuse.
- ◆ Java swing components use the Mediator pattern heavily, notifying a mediator when events occur rather than taking responsibility for updating other components directly.

Discussion of Behavioural Patterns -1

- ◆ One theme is **encapsulating variation**:
 - Strategy encapsulates an algorithm
 - State encapsulates a state-dependent behaviour
 - Mediator encapsulates the protocol between objects
 - Iterator encapsulates the traversal of an aggregate object

Discussion of Behavioural Patterns -2

- ◆ Another theme introduces an object that is always used as **an argument**:
 - Visitor: a visitor object is the argument to the accept operation on the objects it visits.
 - Command: the object represents a request.

Discussion of Behavioural Patterns -3

- ◆ A third theme **decouples** senders and receivers:
 - Command: the command object defines the binding between a sender and a receiver.
 - Observer: the observer interface defines a way to signal changes in subjects that allows multiple observers that can vary at run-time.
 - Mediator: other objects communicate via the mediator.

Discussion of Behavioural Patterns -4

- ◆ Behavioural patterns can also be used with other pattern types, for example:
 - the composite pattern works well with the visitor pattern for performing operations on components.
 - Observer could be used to tie two object structures together.
 - the composition could be created using one of the factory patterns.

Follow-up Reading

- ◆ Larman: Chapter 26 and 36
- ◆ Software Pattern References:
 - E. Gamma et al., Design patterns: elements of reusable object-oriented software, Addison-Wesley, 1995
 - F. Buschmann et al., A system of patterns, Wiley, 1996
 - Steven Metsker, Design Patterns Java Workbook, Addison-Wesley, 2002.
- ◆ Michael Duell, "Non-software examples of software design patterns", *Object Magazine*, Jul 97, p52-57