Behavioral Patterns

Behavioral patterns are concerned with:

- the way classes and objects communicate with each other
- algorithms and the assignment of responsibilities between objects

Behavioral Patterns

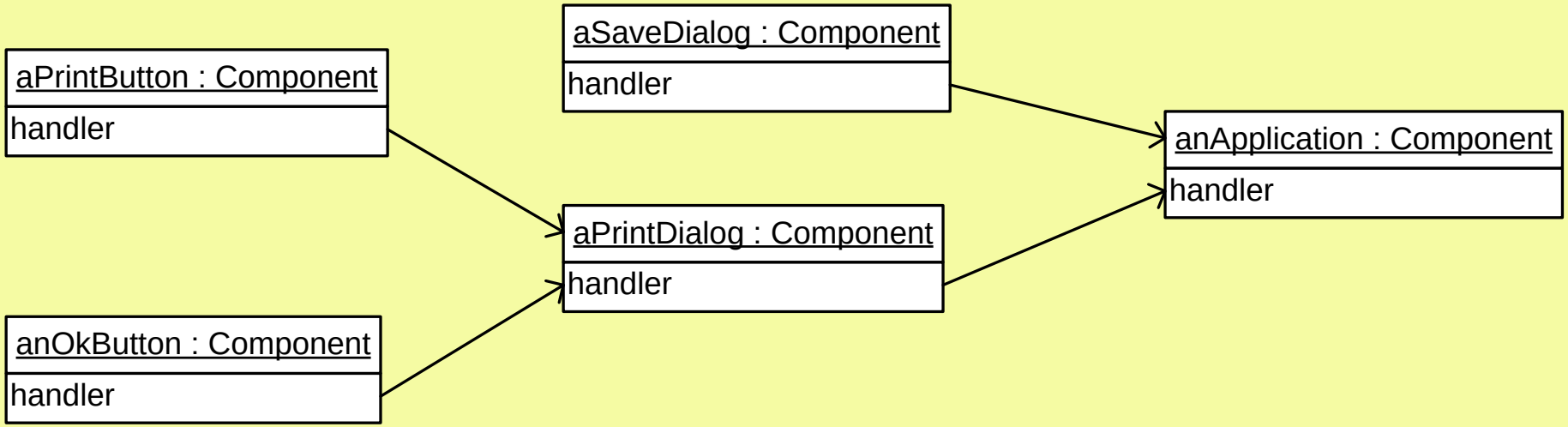
We will study the following ones:

- Chain of Responsibility
- Command
- Observer
- MVC
- State
- Visitor
- Iterator
- Mediator
- Interpreter

Chain of Responsibility: Motivation

- Suppose we want to create a context-sensitive help for a graphical user interface
- When the user clicks on a component:
  - if specific help information exists for the selected component, it appears
  - otherwise, a more general help message about the immediate context appears
- Problem: the object that in the end handles the request isn't known by the object which initiates the request

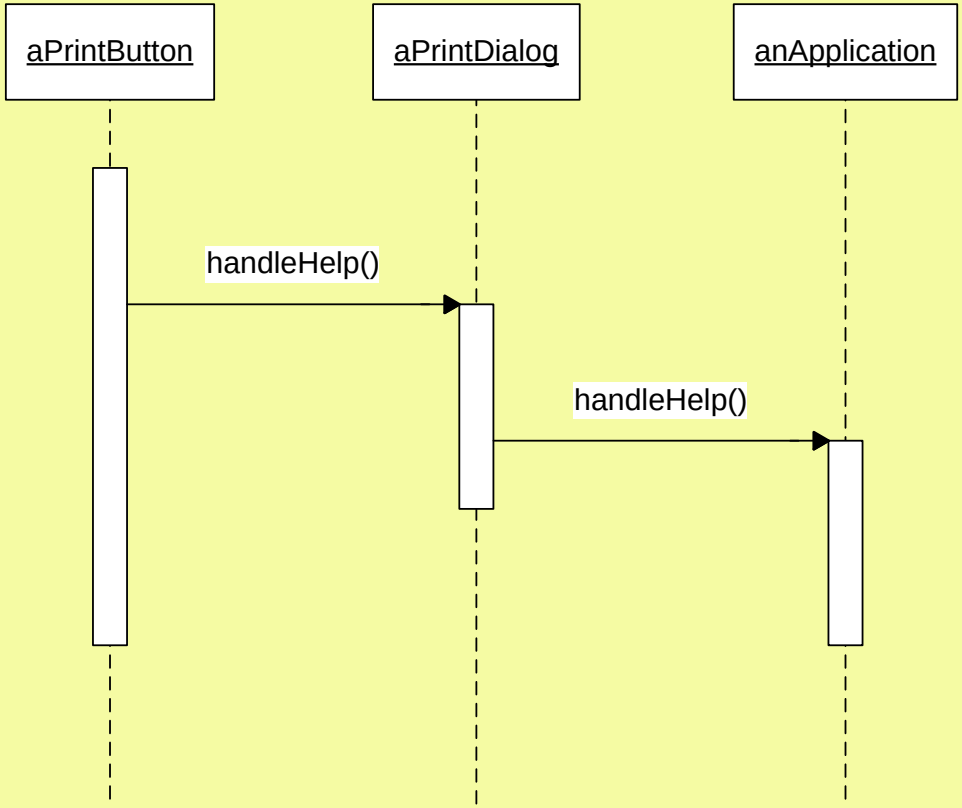
Chain of Responsibility: Motivation



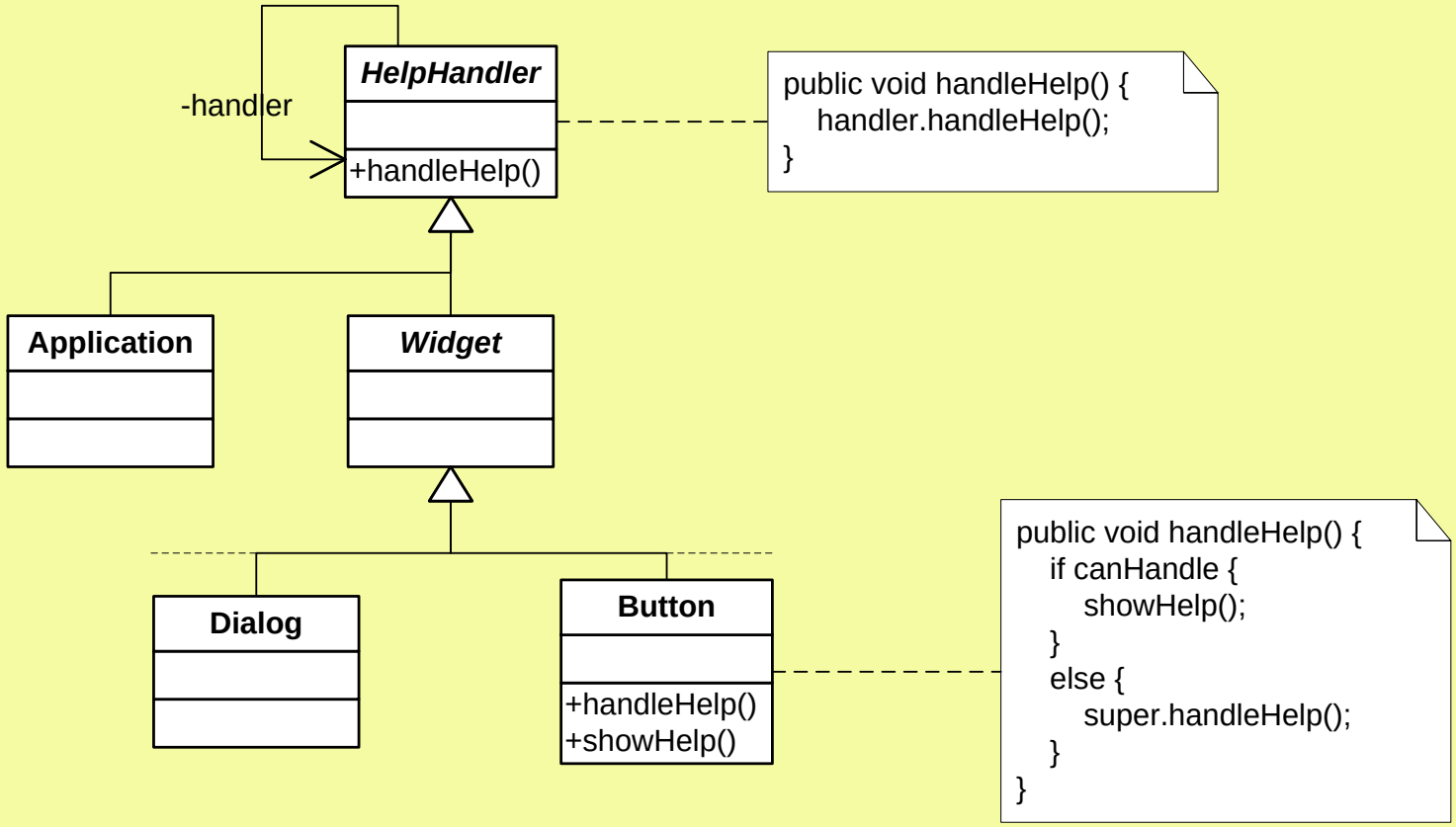
Chain of Responsibility: Intent

- De-couple the sender of a request from its receiver
- The sender doesn't know which receiver treats its request
- The receivers are chained and the request is passed along the chain until an object handles it

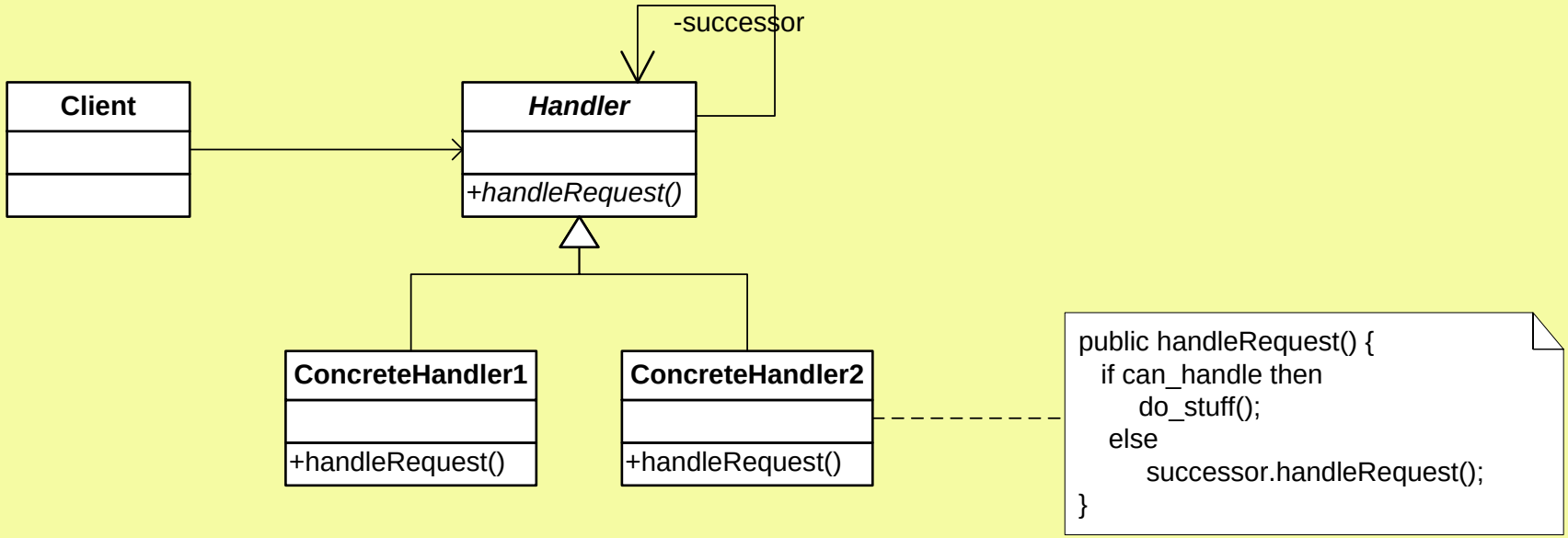
Chain of Responsibility: Motivation



Chain of Responsibilities



Chain of Responsibilities: Structure



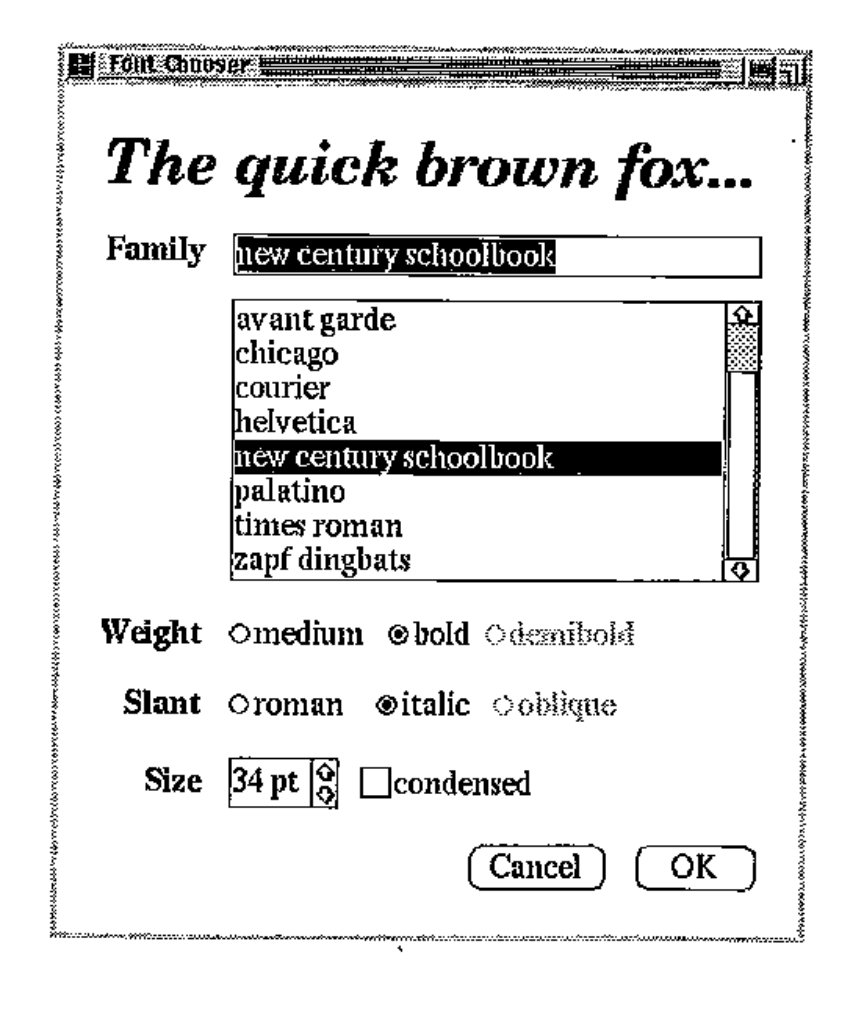
Participants

- Handler: defines the interface for handling requests
- ConcreteHandler: handles requests it is responsible for; forwards other requests to its successor
- Client: initiates a request to a ConcreteHandler object on the chain

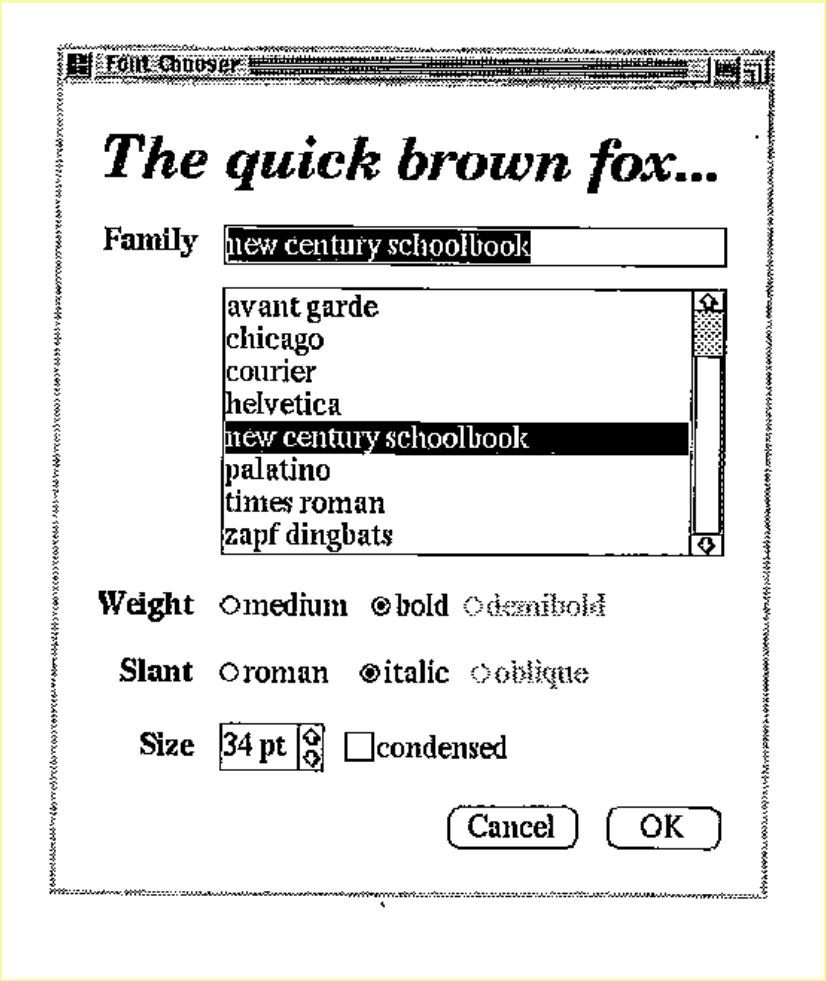
Chain of Responsibility: Consequences

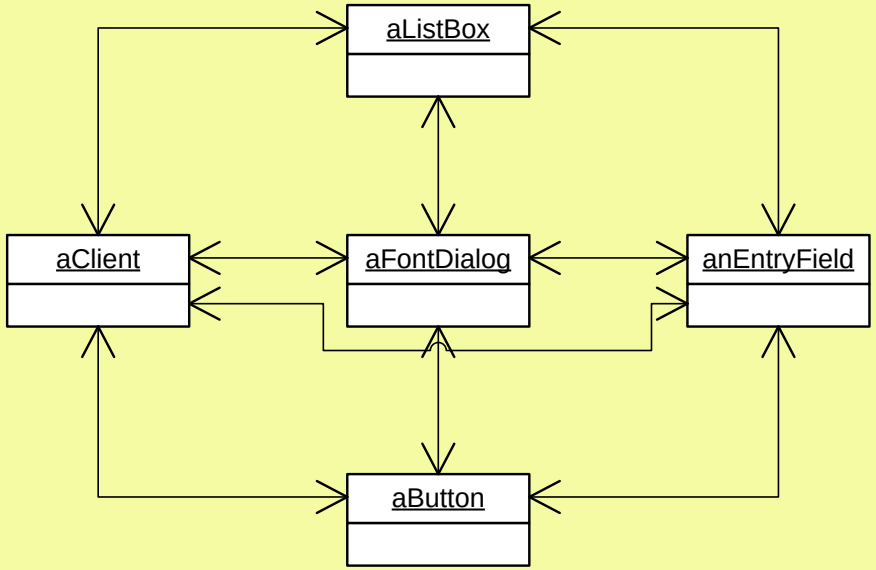
- (+) Reduced coupling:
  - the pattern frees an object from knowing which other object handles the request. Sender and receiver have no explicit knowledge of each other.
  - Instead of objects maintaining references to all candidate receivers, they keep a single reference to their successor.
- (+) Added flexibility in assigning responsibilities
  - You can add or change responsibilities for handling a request by adding to or otherwise changing the chain at run-time
- (-) Receipt isn't guaranteed
  - A request can fall off the end of the chain without ever being handled

Mediator Pattern: Motivation



Mediator Pattern: Motivation (2)

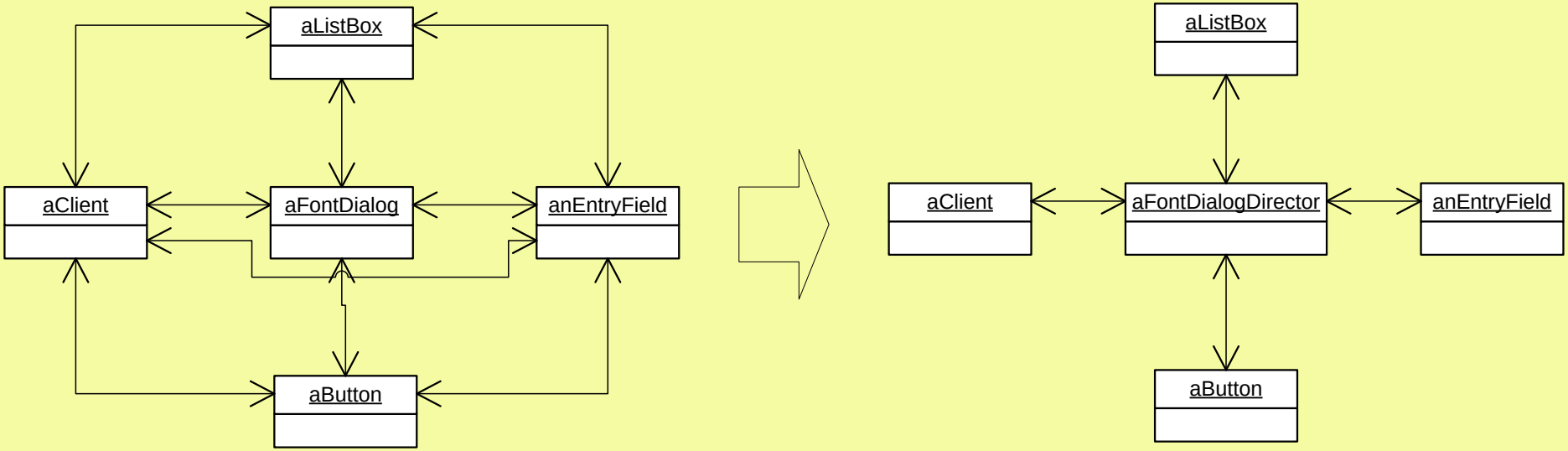




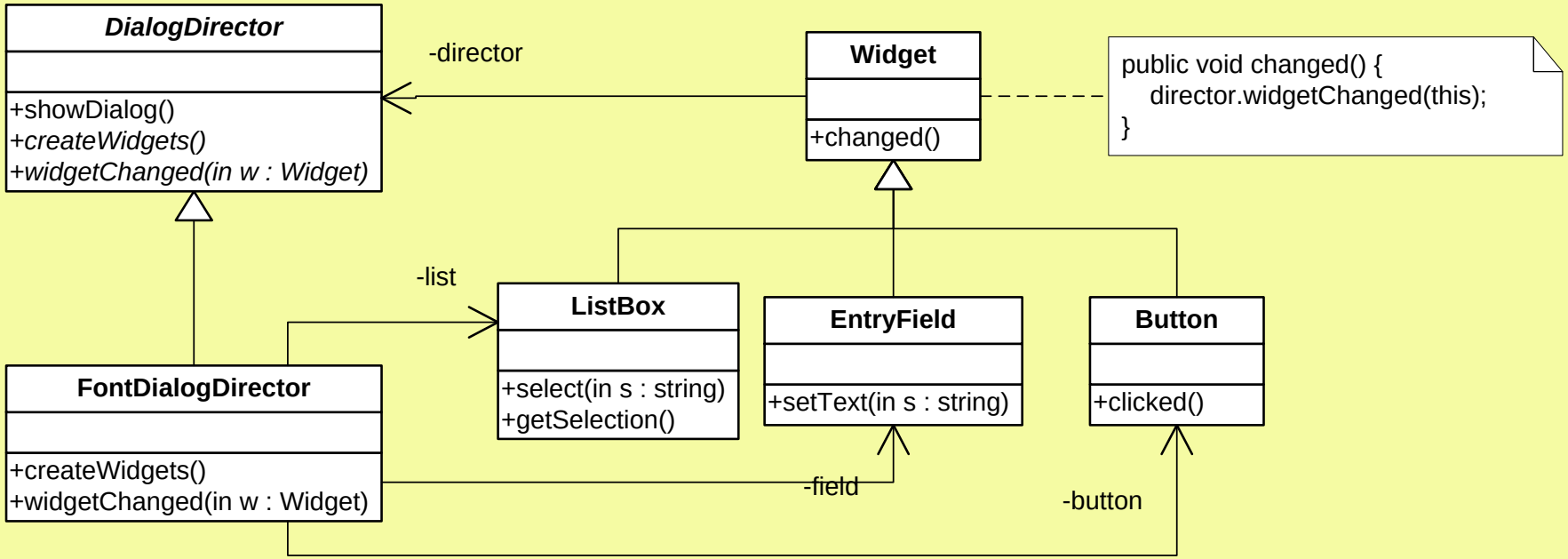
Mediator Pattern: Intent

- Define an object that encapsulates how a set of objects interact
- Mediator promotes loose coupling by keeping objects from referring to each other explicitly
- Their interactions can vary independently

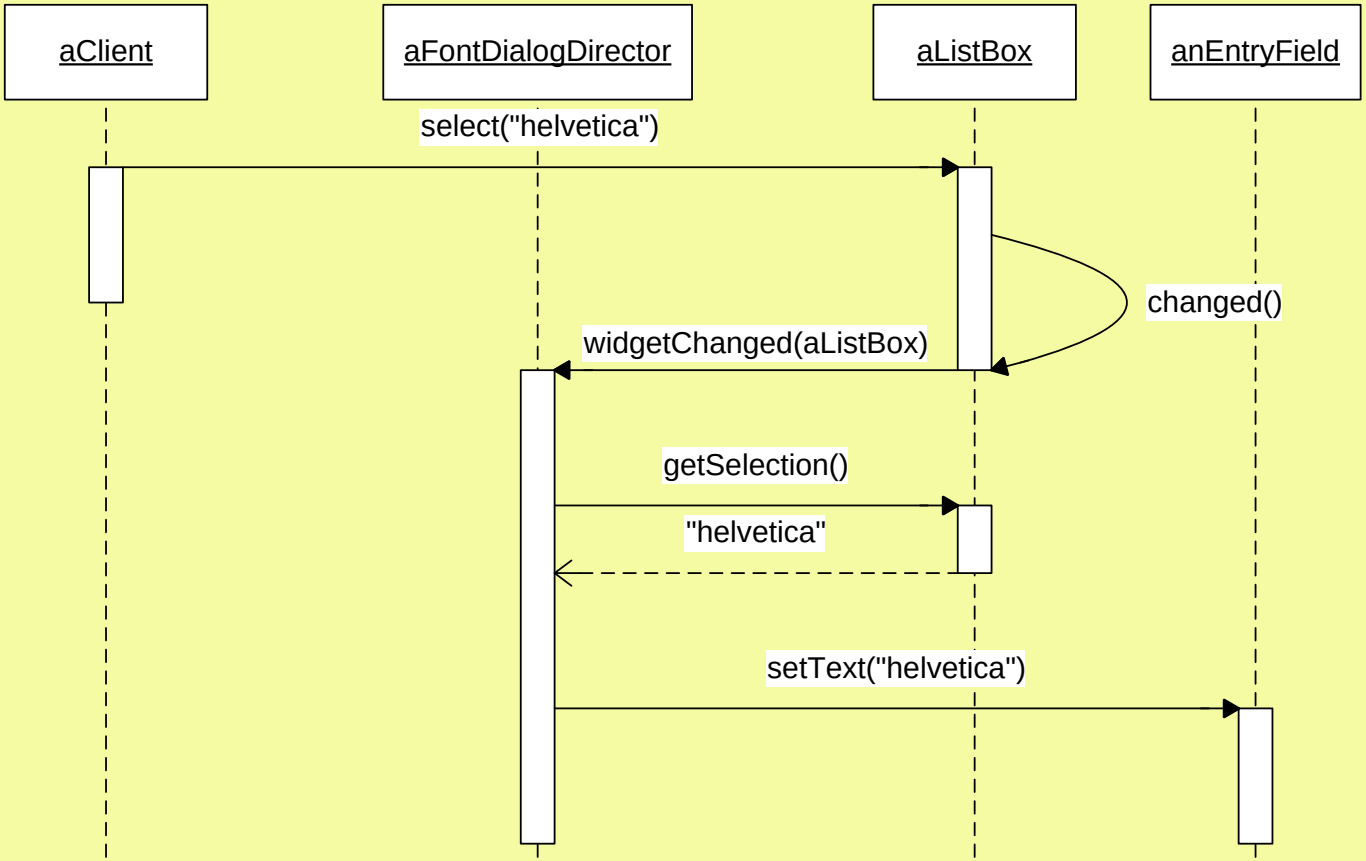
Mediator: Key Idea



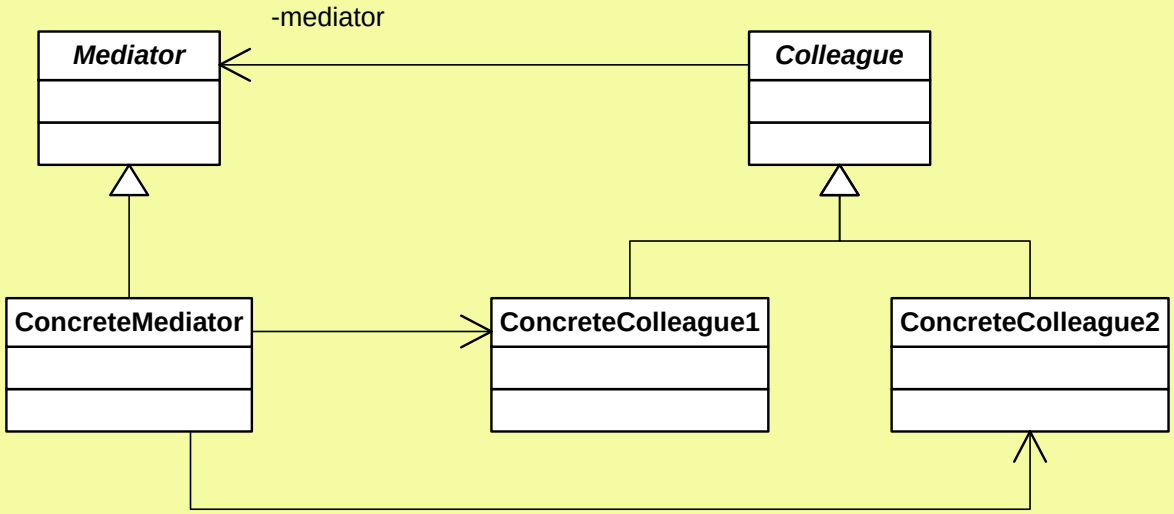
Mediator Pattern at work



Mediator: Interactions



Mediator: structure & participants



- Mediator (DialogDirector):
  - defines an interface for communicating with Colleague objects
- ConcreteMediator (FontDialogDirector)
  - implements cooperative behavior by coordinating Colleague objects
  - knows and maintains its colleagues
- Colleague classes (ListBox, EntryField)
  - each Colleague class knows its Mediator object
  - each Colleague communicates with its mediator whenever it would have otherwise communicated with another colleague

Mediator: consequences

- Mediator pattern limits subclassing by localizing behavior
- It decouples colleagues: they can change and be reused independently
- It simplifies object protocols: many-to-many interactions are replaced by several one-to-many interactions
- It abstracts how objects cooperate
- It centralizes control: the mediator can grow more complex than any individual colleague

Design Patterns