# Mediator Pattern

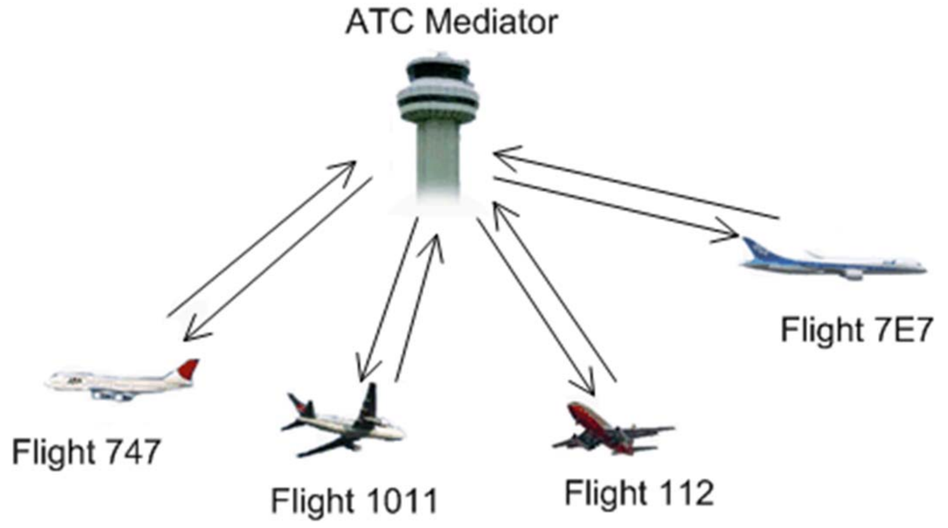
## Contents

## Mediator Pattern

- Purpose
  - Allows loose coupling by encapsulating the way disparate sets of objects interact and communicate with each other.
- Use When
  - Communication between sets of objects is well defined and complex.
  - Too many relationships exist and common point of control or communication is needed.

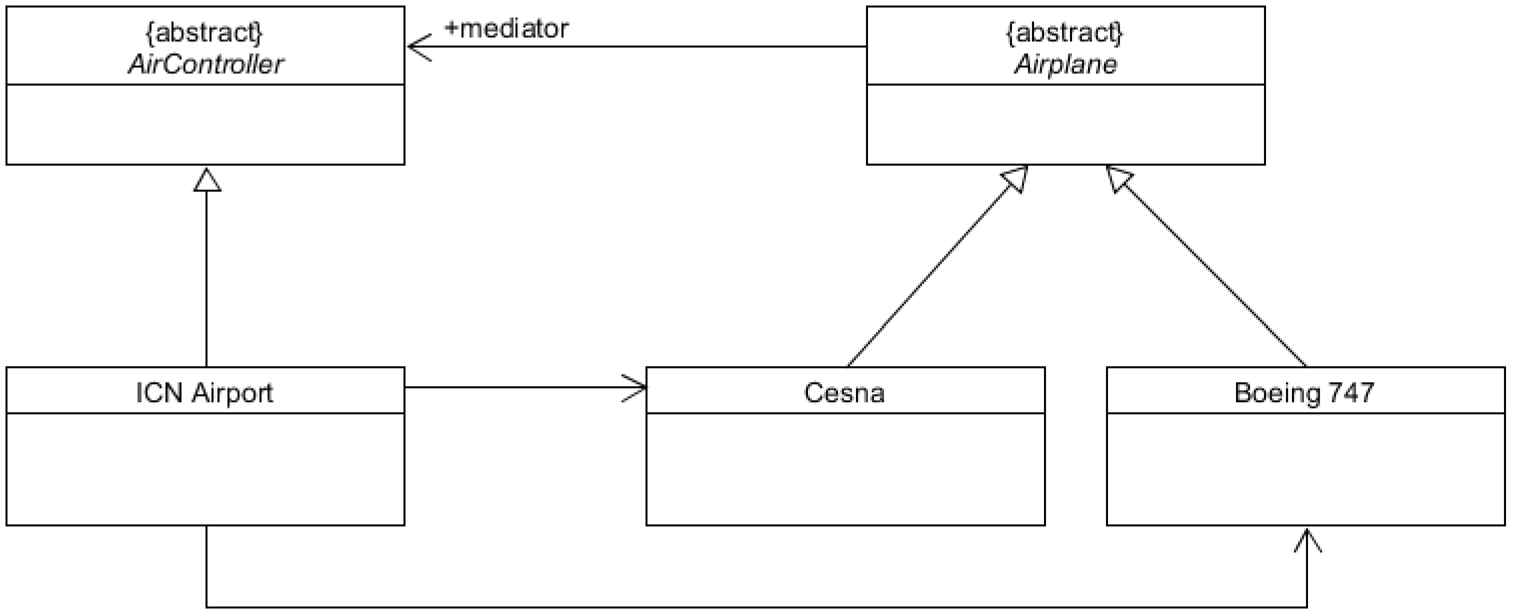
## Battling Class Complexity

- Consider an air traffic controller
  - Many planes circle an airport
  - If they communicated with each other the skies above an airport would be a chaos
  - Accepted solution: have the planes communicate directly with an air traffic controller tower for permission to land
  - The planes do not even have to know about each other
  - This information is kept with the tower

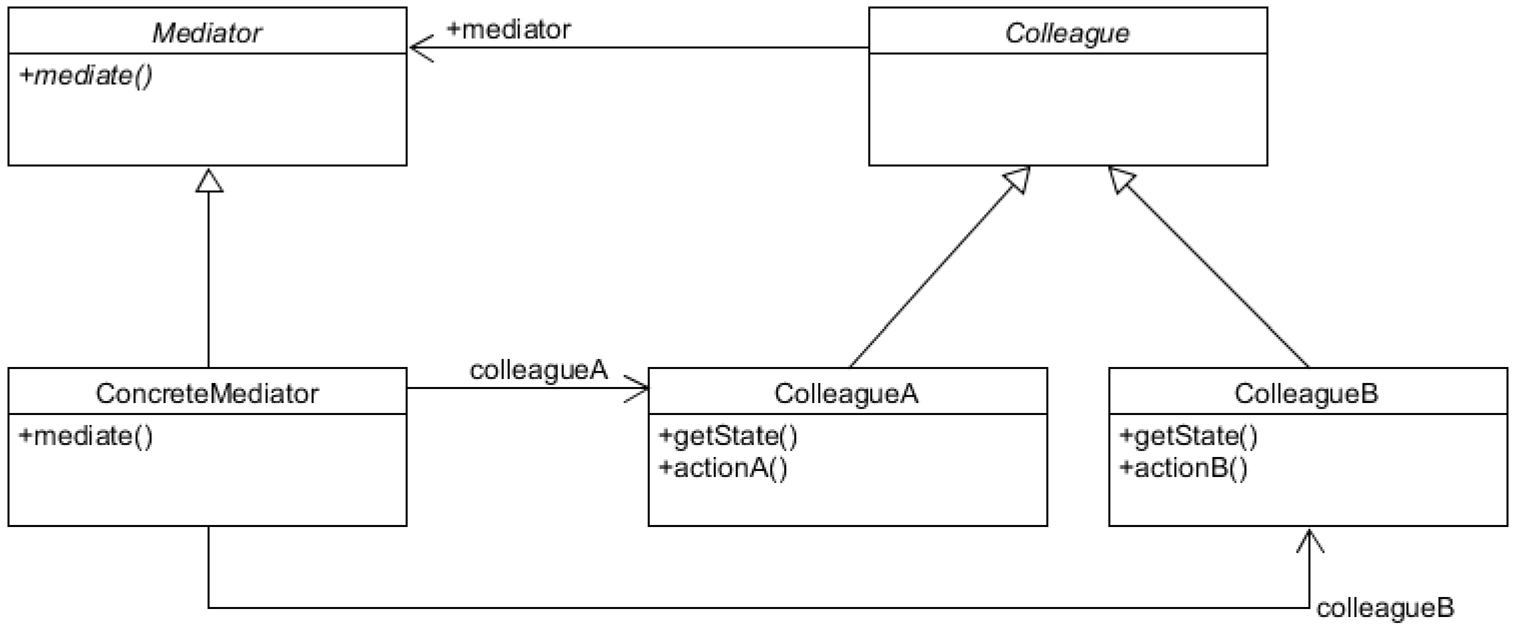


## Battling Class Complexity

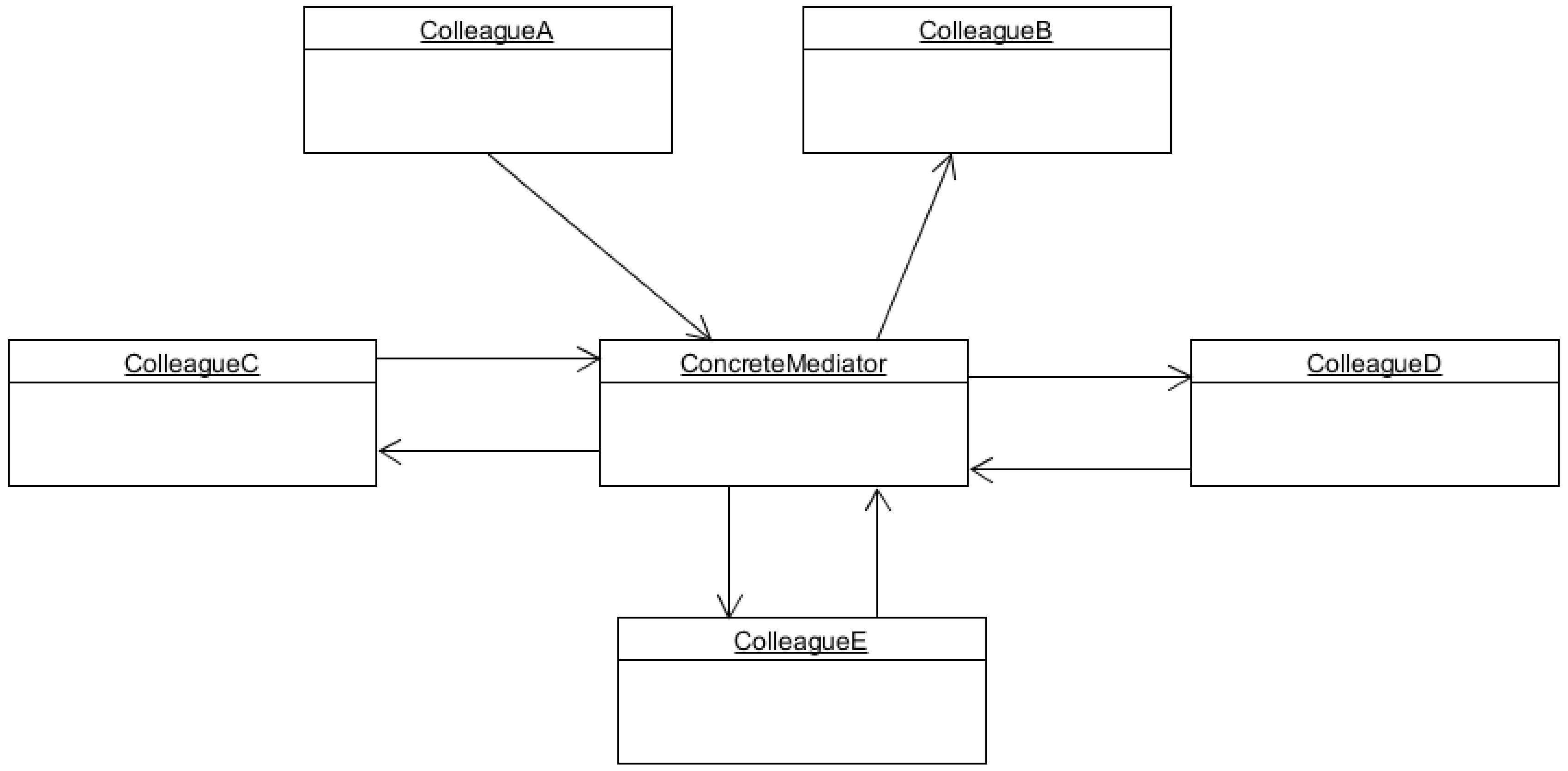
- Let's abstract this to classes and their objects in a program
  - When objects are allowed to communicate directly with each other, then they become too tightly coupled
  - When one object wishes to send a message to another then we need the equivalent of an air traffic controller to forward the message to the recipient
  - Keep the dispatching information inside the new controller
  - Call this coordinating object a mediator



# Class Diagram for Mediator



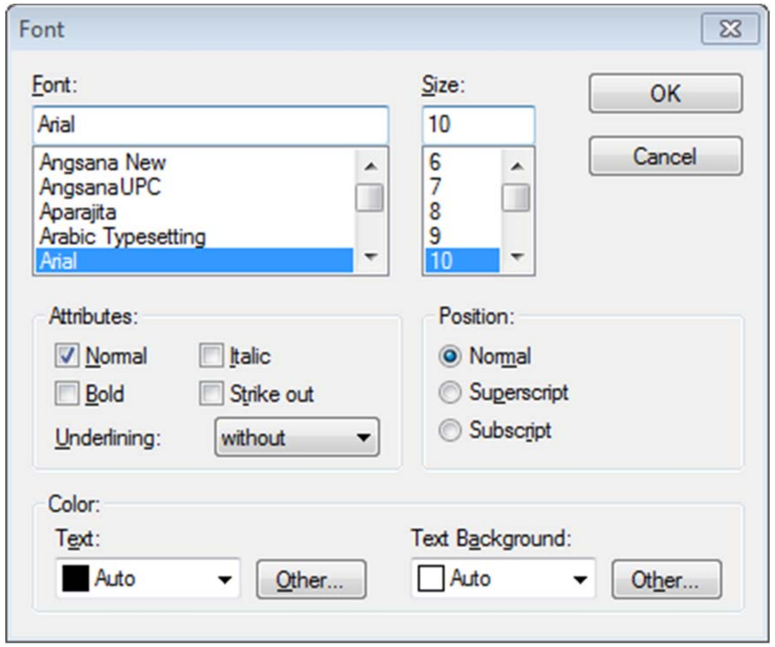
## Object Diagram for Mediator



## Mediator Pattern

- Encapsulates interconnects between objects into Mediator
  - communications hub
  - Responsible for coordinating and controlling colleague interaction
- Promotes loose coupling between classes
  - By preventing from referring to each other explicitly
  - Mediator is commonly used to coordinate related GUI components
- (-) mediators are hardly ever reusable
- (+) easy to understand the flow of communication

# Mediator Example: FontDialog



Font: Arial

Angsana New
AngsanaUPC
Aparajita
Arabic Typesetting
Arial

Size: 10

6
7
8
9
10

Attributes:
[✓ Normal] Italic
Bold Strike out
Underlining: without

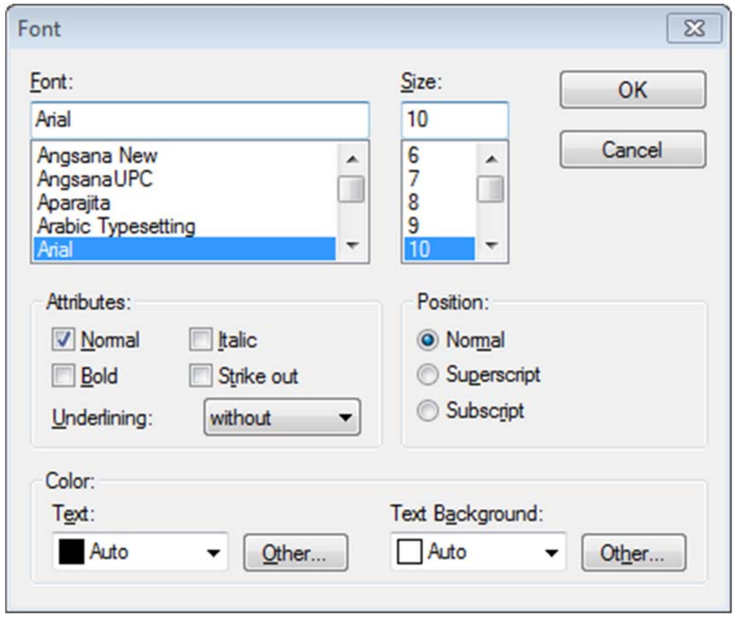
Position:
[✓ Normal]
Superscript
Subscript

Color:
Text: ■ Auto
Text Background: □ Auto

Often there are dependencies between the widgets in the dialog. For example, a button gets disabled when a certain entry field is empty. Selecting an entry in a list of choices called a **list box** might change the contents of an entry field.

Conversely, typing text into the entry field might automatically select one or more corresponding entries in the list box. Once text appears in the entry field, other buttons may become enabled that let the user do something with the text, such as changing or deleting the thing to which it refers.

# Mediator Example: FontDialog



Font

Font: Arial

Angsana New
AngsanaUPC
Aparajita
Arabic Typesetting
Arial

Size: 10

6
7
8
9
10

Attributes:

[✓ Normal] Italic
Bold Strike out

Underlining: without

Position:

[✓ Normal]
Superscript
Subscript

Color:

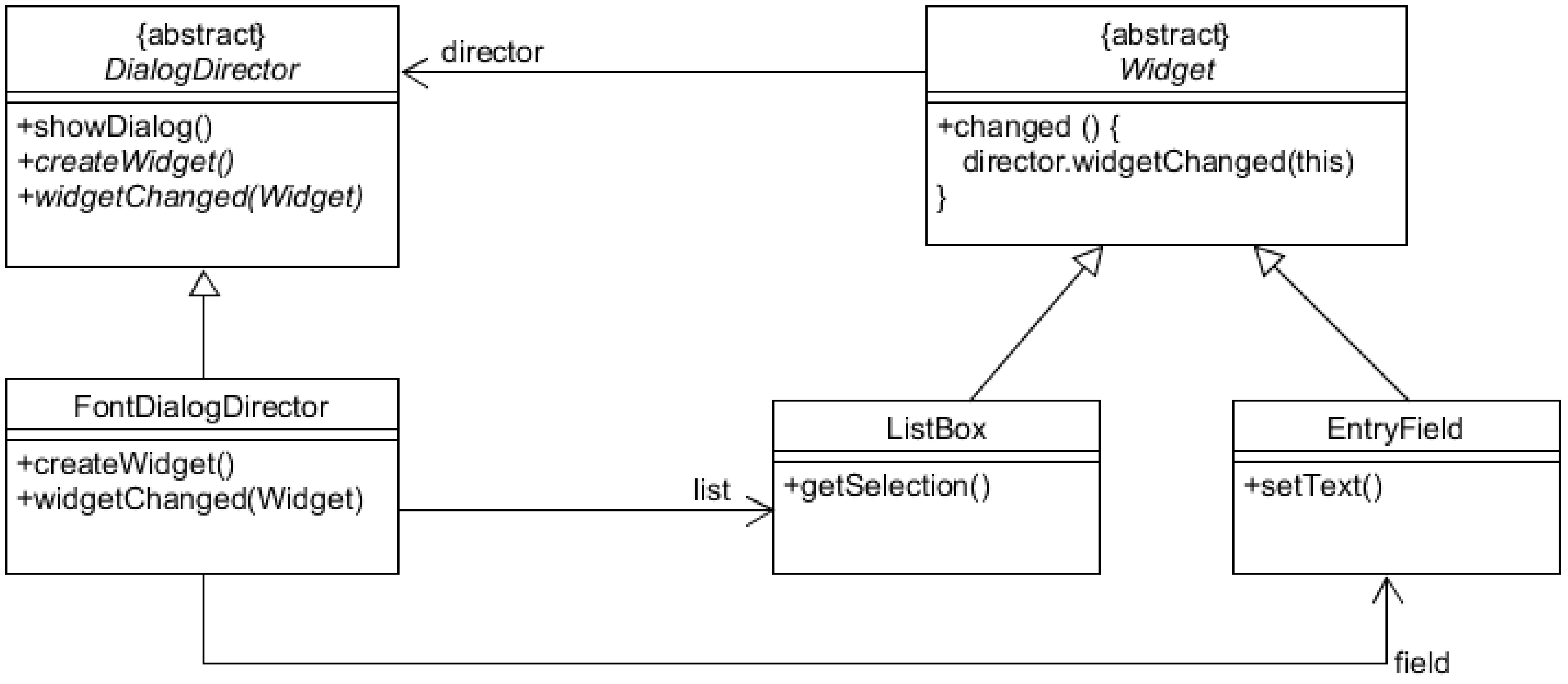
Text: Auto

Text Background: Auto

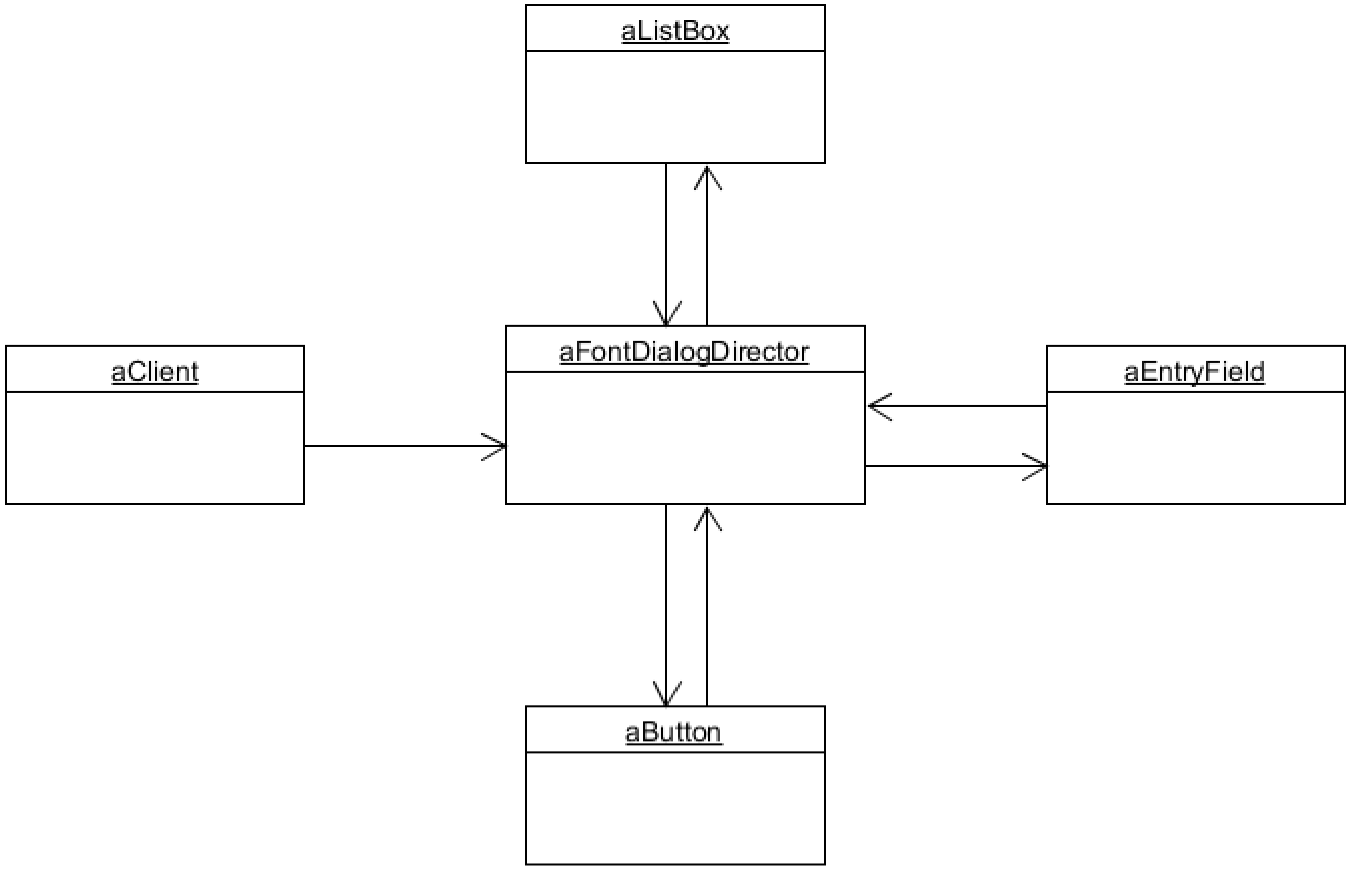
Different dialog boxes will have different dependencies between widgets. So even though dialogs display the same kinds of widgets, they can't simply reuse stock widget classes; they have to be customized to reflect dialog-specific dependencies. Customizing them individually by subclassing will be tedious, since many classes are involved.

You can avoid these problems by encapsulating collective behavior in a separate **mediator** object.

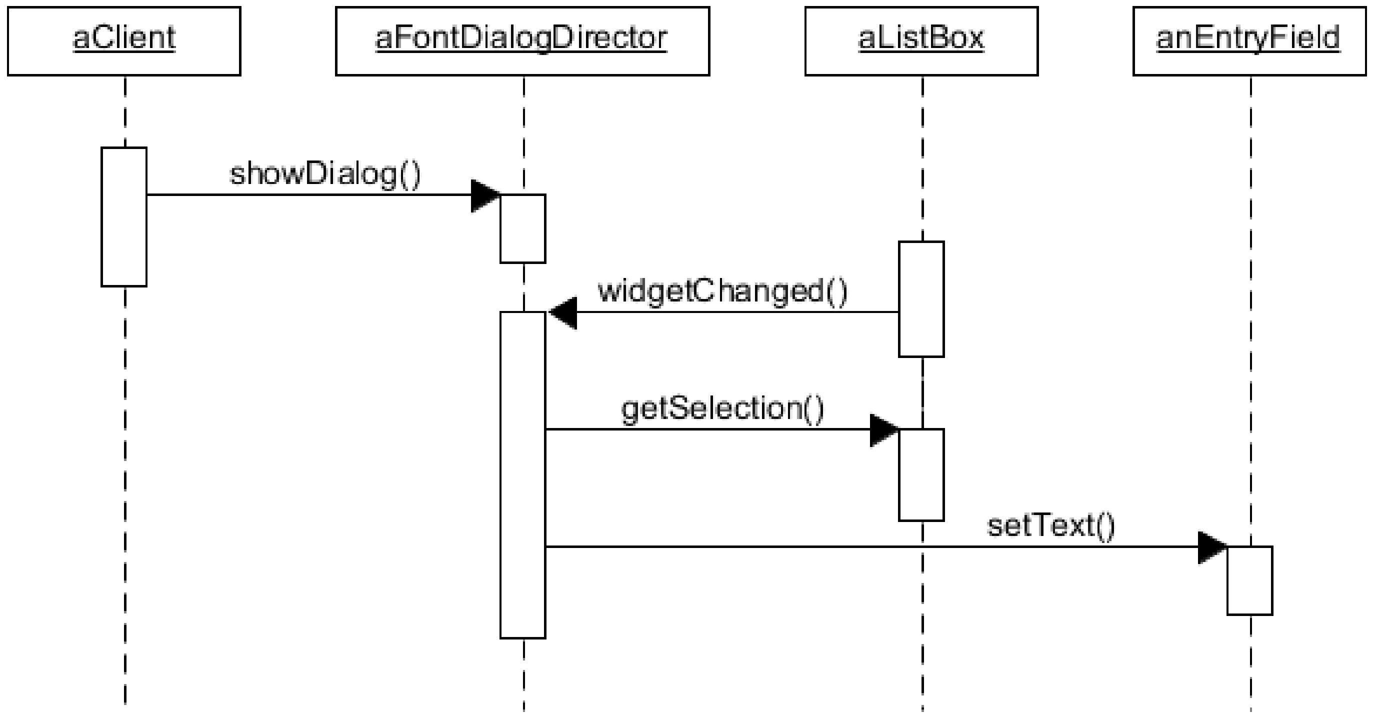
## Structure of FontDialog



# Object Diagram of FontDialog



## Sequence diagram of FontDialog



1. The list box tells its director that it's changed.
2. The director gets the selection from the list box.
3. The director passes the selection to the entry field.
4. Now that the entry field contains some text, the director enables button(s) for initiating an action (e.g., "demibold," "oblique").

## Related Patterns and Summary

- Observer
  - Communication distributed by using observer and subject objects
  - reusable
  - can be hard to understand the multiple flows of communication

- Mediator
  - Mediator encapsulates the communication
  - mediators are hardly ever reusable
  - easy to understand the flow of communication