Design Patterns

MSc in Computer Science

Produced by

Eamonn de Leastar (edeleastar@wit.ie)

Department of Computing, Maths & Physics
Waterford Institute of Technology

http://www.wit.ie

http://elearning.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit

Mediator

Behavioural Pattern

Intent

- Define an object that encapsulates how a set of objects interact.

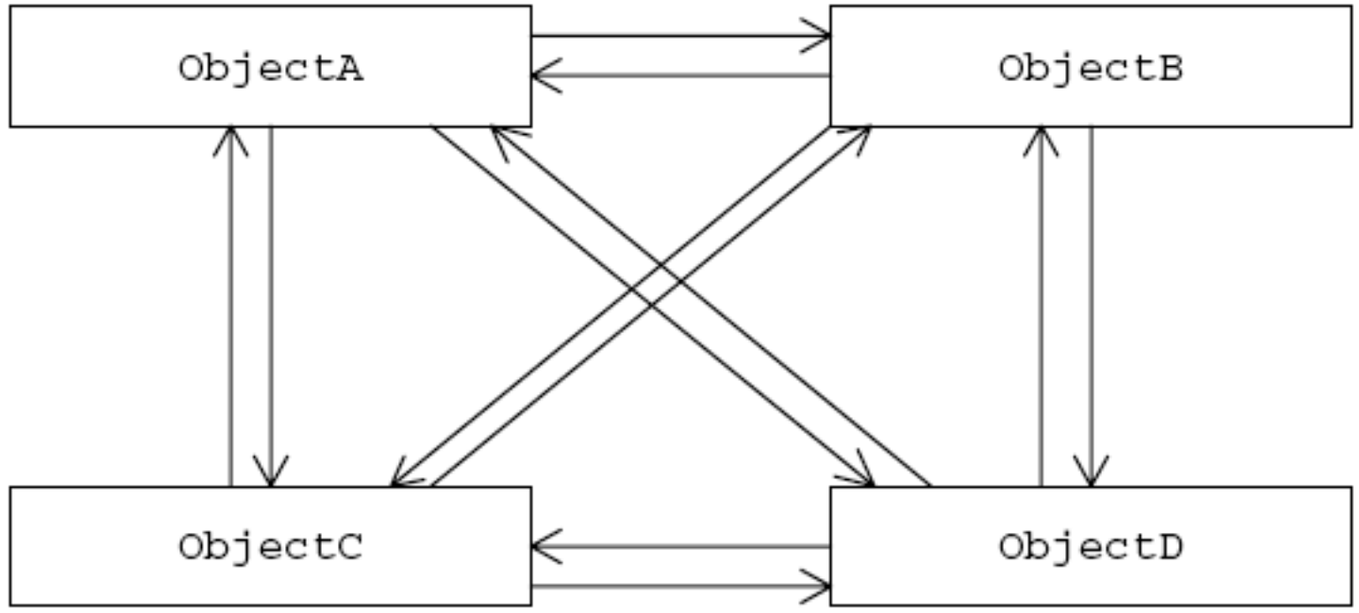
- Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.

Motivation (1)

- Object-oriented design encourages the distribution of behavior among objects.

- Such distribution can result in an object structure with many connections between objects; in the worst case, every object ends up knowing about every other.

- Though partitioning a system into many objects generally enhances reusability, proliferating interconnections tend to reduce it again.



Motivation (2)

- Lots of interconnections make it less likely that an object can work without the support of others—the system acts as though it were monolithic.

- Moreover, it can be difficult to change the system's behavior in any significant way, since behavior is distributed among many objects.

- As a result, you may be forced to define many subclasses to customize the system's behavior.

Motivation (3) (Holub example)

- The method ask() displays the small window.

- It accepts an answer and, if enter pressed, the window shuts down, and ask(…) returns the entered string.

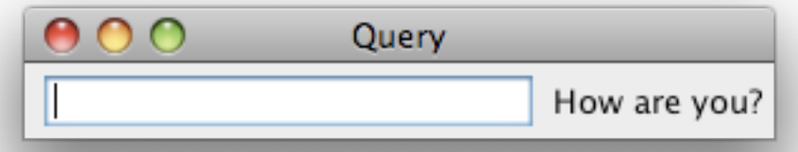
- If the window is closed then ask(..) returns an empty string.

- The code encapsulates a complex interaction with the GUI subsystem, but the user exercises all this complexity by doing a simple thing.

- The details are all hidden. Moreover, code that uses Query is now considerably simplified.

```
public static void main(String[] args)
{
 SwingQuery query = new SwingQuery();

 String result = query.ask("How are you?");
 System.out.println(result);
}
```

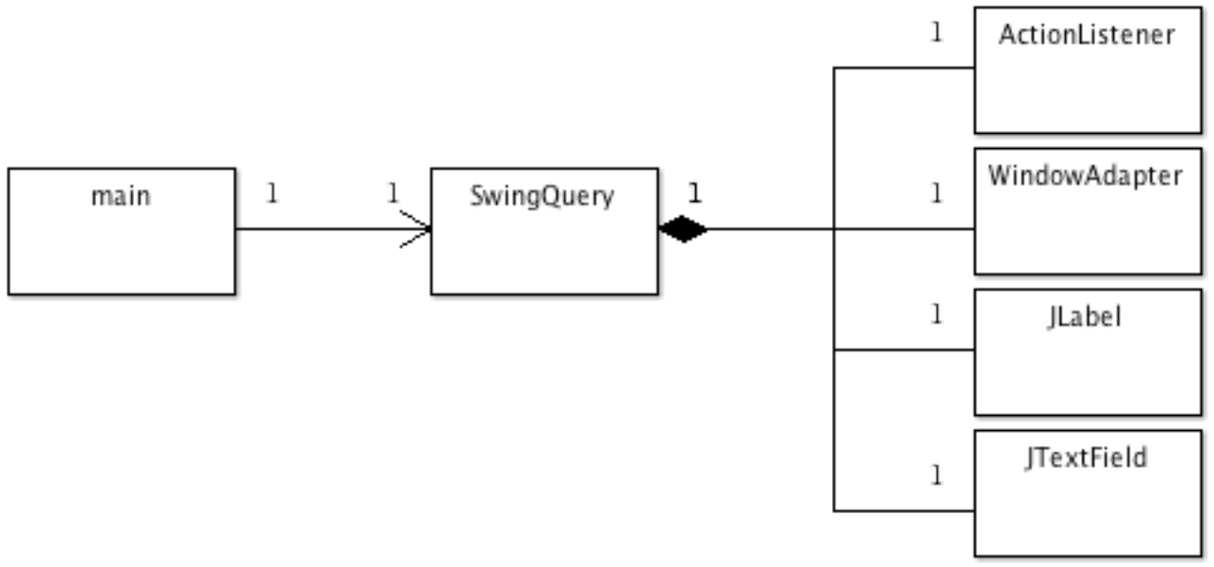


```
class SwingQuery implements Query
{
  public String ask(String question)
  {
    final Object done = new Object();
    final Object init = new Object();
    final JFrame frame = new JFrame("Query");
    final JTextField answer = new JTextField();
    answer.setPreferredSize(new Dimension(200, 20));
    frame.getContentPane().setLayout(new FlowLayout());
    frame.getContentPane().add(answer);
    frame.getContentPane().add(new JLabel(question));
    answer.addActionListener
     (new ActionListener()
     {
       public void actionPerformed(ActionEvent e)
       {
         synchronized (init)
         {
           synchronized (done)
           {
             frame.dispose();
             done.notify();
           }
         }
       }
     });
...
```

```
  frame.addWindowListener
     (new WindowAdapter()
     {
       public void windowClosing(WindowEvent e)
       {
         synchronized (init)
         {
           synchronized (done)
           {
             frame.dispose();
             answer.setText("");
             done.notify();
           }
         }
       }
     });
    synchronized (done)
    {
      synchronized (init)
      {
        frame.pack();
        frame.setVisible(true);
      }
      try
      {
        done.wait();
      }
      catch (InterruptedException e)
      {
      }
    }
    return answer.getText();
  }
}
```

SwingQuery Structure

- SwingQuery hides
  - ActionListener
  - WindowAdapter
  - JLabel
  - JTextField



- + multi threading issues associated with Swing

Applicability

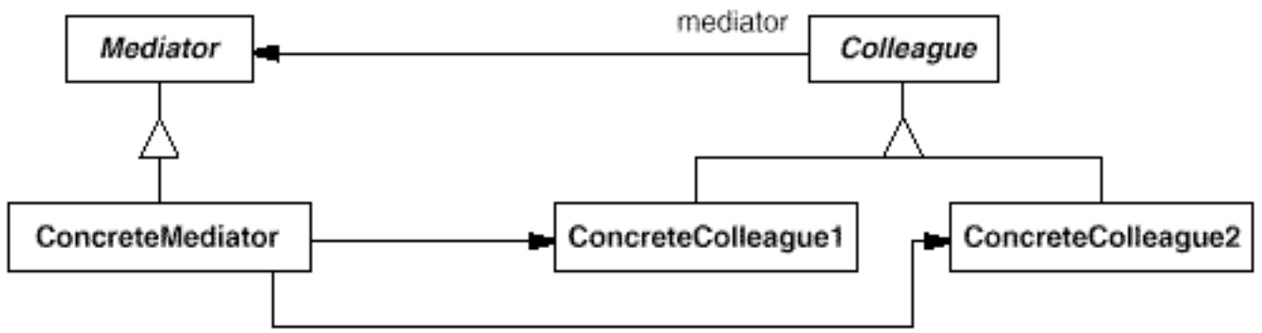
- A set of objects communicate in well-defined but complex ways. The resulting interdependencies are unstructured and difficult to understand.

- Reusing an object is difficult because it refers to and communicates with many other objects.

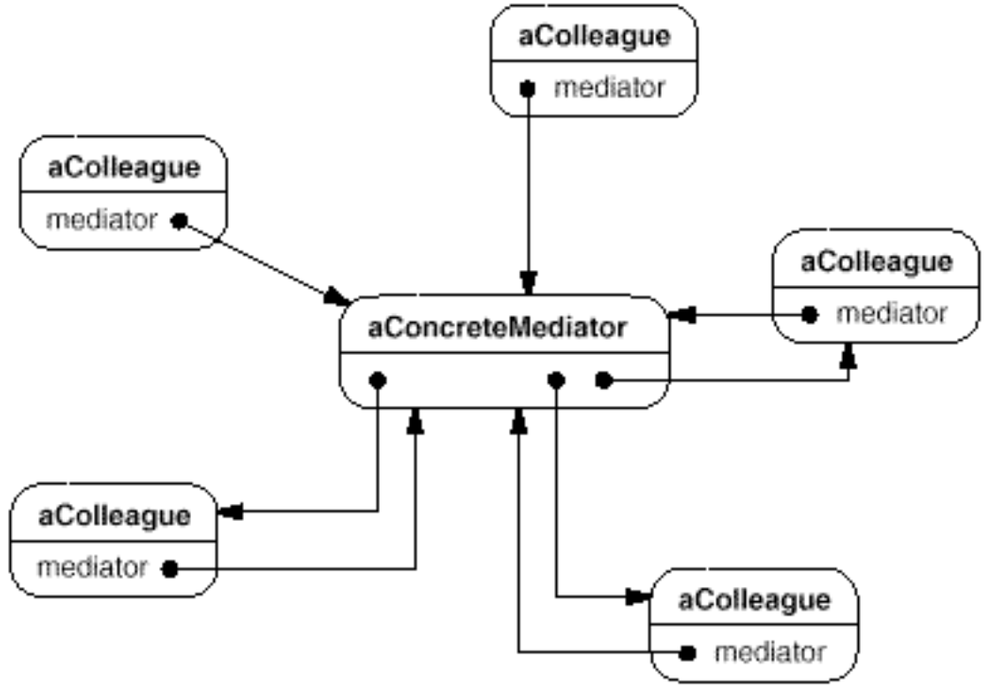
- A behavior that's distributed between several classes should be customizable without a lot of subclassing.

Structure

- Class

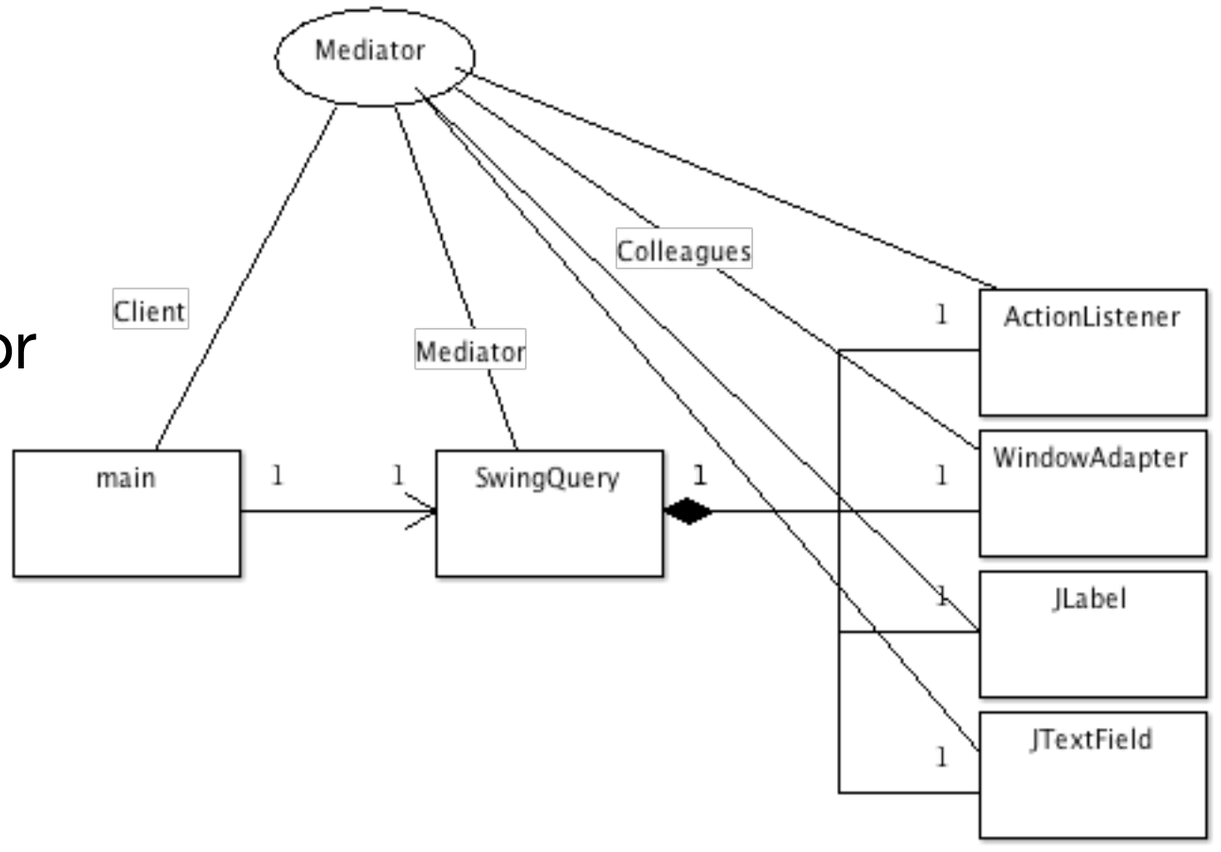


- Object



Participants

- Mediator (Not used)
  - defines an interface for communicating with Colleague objects.
- ConcreteMediator (SwingQuery)
  - implements cooperative behavior by coordinating Colleague objects.
  - knows and maintains its colleagues.
- Colleague classes
  - Subsystem being mediated.
  - (JTextField, JLabel, WindowAdapter, ActionListener)



Collaborations

- Colleagues send and receive requests from a Mediator object.
- The mediator implements the cooperative behavior by routing requests between the appropriate colleague(s).

Consequences (1)

- It limits subclassing. A mediator localizes behavior that otherwise would be distributed among several objects. Changing this behavior requires subclassing Mediator only; Colleague classes can be reused as is.

- It decouples colleagues. A mediator promotes loose coupling between colleagues. You can vary and reuse Colleague and Mediator classes independently.

- It simplifies object protocols. A mediator replaces many-to-many interactions with one-to-many interactions between the mediator and its colleagues. One-to-many relationships are easier to understand, maintain, and extend.

Consequences (2)

- It abstracts how objects cooperate. Making mediation an independent concept and encapsulating it in an object lets you focus on how objects interact apart from their individual behavior. That can help clarify how objects interact in a system.

- It centralizes control. The Mediator pattern trades complexity of interaction for complexity in the mediator. Because a mediator encapsulates protocols, it can become more complex than any individual colleague. This can make the mediator itself a monolith that's hard to maintain.