Mediator in C++

Back to **Mediator** description

Mediator design pattern demo

Discussion. Though partitioning a system into many objects generally enhances reusability, proliferating interconnections tend to reduce it again. You can avoid this problem by encapsulating the interconnections (i.e. the collective behavior) in a separate "mediator" object. A mediator is responsible for controlling and coordinating the interactions of a group of objects.

In this example, the dialog box object is functioning as the mediator. Child widgets of the dialog box do not know, or care, who their siblings are. Whenever a simulated user interaction occurs in a child widget `Widget::changed()`, the widget does nothing except "delegate" that event to its parent dialog box `mediator->widgetChanged(this)`.

`FileSelectionDialog::widgetChanged()` encapsulates all collective behavior for the dialog box (it serves as the hub of communication). The user may choose to "interact" with a simulated: filter edit field, directories list, files list, or selection edit field.

```cpp
#include <iostream.h>

class FileSelectionDialog;

class Widget
{
  public:
    Widget(FileSelectionDialog *mediator, char *name)
    {
        _mediator = mediator;
        strcpy(_name, name);
    }
    virtual void changed();
    virtual void updateWidget() = 0;
    virtual void queryWidget() = 0;
  protected:
    char _name[20];
  private:
    FileSelectionDialog *_mediator;
};

class List: public Widget
{
  public:
    List(FileSelectionDialog *dir, char *name): Widget(dir, name){}
    void queryWidget()
    {
        cout << "   " << _name << " list queried" << endl;
    }
    void updateWidget()
    {
        cout << "   " << _name << " list updated" << endl;
    }
};

class Edit: public Widget
{
  public:
    Edit(FileSelectionDialog *dir, char *name): Widget(dir, name){}
    void queryWidget()
    {
        cout << "   " << _name << " edit queried" << endl;
    }
    void updateWidget()
    {
        cout << "   " << _name << " edit updated" << endl;
    }
};
```

```
class FileSelectionDialog
{
  public:
    enum Widgets
    {
        FilterEdit, DirList, FileList, SelectionEdit
    };
    FileSelectionDialog()
    {
        _components[FilterEdit] = new Edit(this, "filter");
        _components[DirList] = new List(this, "dir");
        _components[FileList] = new List(this, "file");
        _components[SelectionEdit] = new Edit(this, "selection");
    }
    virtual ~FileSelectionDialog();
    void handleEvent(int which)
    {
        _components[which]->changed();
    }
    virtual void widgetChanged(Widget *theChangedWidget)
    {
        if (theChangedWidget == _components[FilterEdit])
        {
            _components[FilterEdit]->queryWidget();
            _components[DirList]->updateWidget();
            _components[FileList]->updateWidget();
            _components[SelectionEdit]->updateWidget();
        }
        else if (theChangedWidget == _components[DirList])
        {
            _components[DirList]->queryWidget();
            _components[FileList]->updateWidget();
            _components[FilterEdit]->updateWidget();
            _components[SelectionEdit]->updateWidget();
        }
        else if (theChangedWidget == _components[FileList])
        {
            _components[FileList]->queryWidget();
            _components[SelectionEdit]->updateWidget();
        }
        else if (theChangedWidget == _components[SelectionEdit])
        {
            _components[SelectionEdit]->queryWidget();
            cout << "   file opened" << endl;
        }
    }
  private:
    Widget *_components[4];
```

```
};

FileSelectionDialog::~FileSelectionDialog()
{
  for (int i = 0; i < 4; i++)
    delete _components[i];
}

void Widget::changed()
{
  _mediator->widgetChanged(this);
}

int main()
{
  FileSelectionDialog fileDialog;
  int i;

  cout << "Exit[0], Filter[1], Dir[2], File[3], Selection[4]: ";
  cin >> i;

  while (i)
  {
    fileDialog.handleEvent(i - 1);
    cout << "Exit[0], Filter[1], Dir[2], File[3], Selection[4]: ";
    cin >> i;
  }
}
```

Output

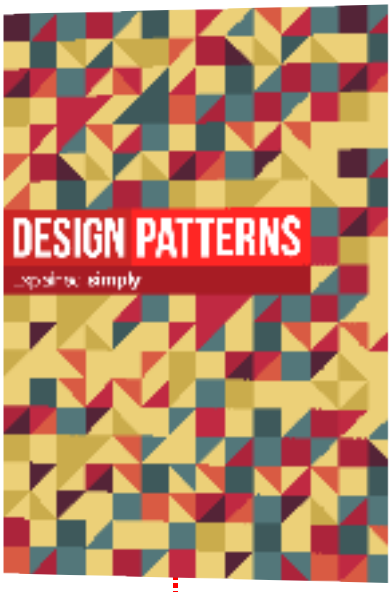
```
Exit[0], Filter[1], Dir[2], File[3], Selection[4]: 1
   filter edit queried
   dir list updated
   file list updated
   selection edit updated
Exit[0], Filter[1], Dir[2], File[3], Selection[4]: 2
   dir list queried
   file list updated
   filter edit updated
   selection edit updated
Exit[0], Filter[1], Dir[2], File[3], Selection[4]: 3
   file list queried
   selection edit updated
Exit[0], Filter[1], Dir[2], File[3], Selection[4]: 4
   selection edit queried
   file opened
Exit[0], Filter[1], Dir[2], File[3], Selection[4]: 3
   file list queried
   selection edit updated
```

Read next

This article is taken from our book **Design Patterns Explained Simply**.

All of the design patterns are compiled there. The book is written in clear, simple language that makes it easy to read and understand (just like this article).

We distribute it in PDF & EPUB formats so you can get it onto your iPad, Kindle, or other portable device immediately after your purchase.



Learn more

Code examples

Java	**Mediator in Java**	
C++	**Mediator in C++**	**Mediator in C++: Before and after**
PHP	**Mediator in PHP**	
Delphi	**Mediator in Delphi**	

Design Patterns
AntiPatterns
Refactoring
UML

My account
Forum
Contact us
About us