Supplement to

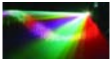
The Design and Implementation of Multimedia Software

The Observer Pattern

Prof. David Bernstein

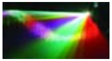
James Madison University

users.cs.jmu.edu/bernstdh



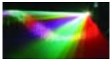
Motivation

- Objects often need to communicate with each other
- Traditional message passing techniques tend to couple objects too tightly



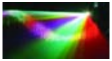
An Example

A security (e.g., stocks, futures, options) trading application in which there is a `TickReader` that reads “tick by tick” pricing information (e.g., from the Internet) and sends each `Tick` to a `TickWriter` that saves the information in a file and to a `TickerTape` that displays the information on a screen.



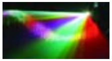
An Example - A Bad Design

Think of the `TickReader` as actively adding ticks to the `TickWriter` and `TickerTape`.



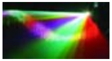
Problems with this Design

- Because `TickReader` calls the `TickWriter` and `TickerTape` it is not very re-usable
- Every `TickReader` must have an associated `TickWriter` and `TickerTape`



An Example - A Better Design

Think of the `TickWriter` and `TickerTape` as passively listening for "ticks". Then, a `TickReader` need only have a list of (zero or more) `TickListener` objects that it will inform.



The Observer Pattern

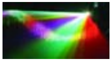
- Intent:

  Define a one-to-many dependency between objects so that when one object changes state, all of its dependents are notified

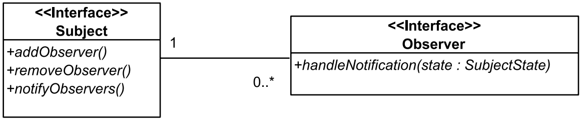
- Participants:

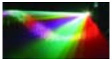
  A subject

  Some observers

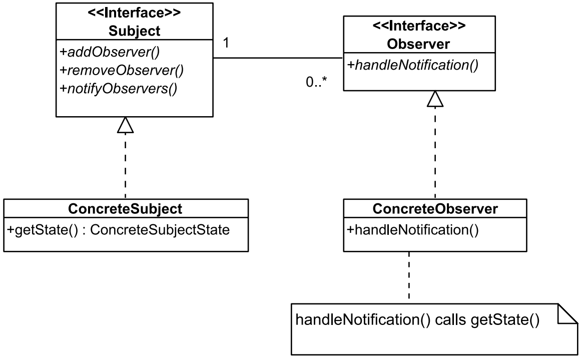


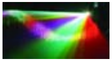
One Implementation





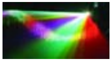
Another Implementation





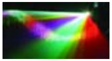
Issues Related to the Choice of Collection

- Frequency of notifications vs. modifications (compare `Hashtable` or `HashMap` with `Vector` or `ArrayList`)
- Need for concurrent notifications and modifications (think about `CopyOnWriteArrayList`)



Other Terminology

- Listener
- Publish-Subscribe



A Complete Example

- A Silly Text Processor:

    Counts the number of words that start with an uppercase letter

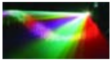
    Save the lines to a file

    Shows the progress (e.g., then number of lines processed)

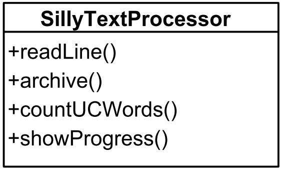
- Some Observations:

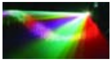
    This is not going to make us any money

    We can use it to explore different designs

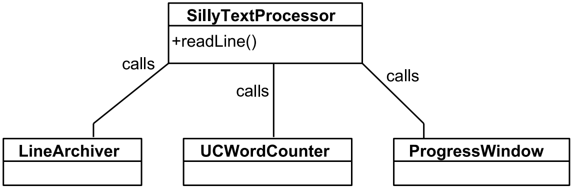


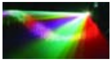
A Design that is Not Cohesive



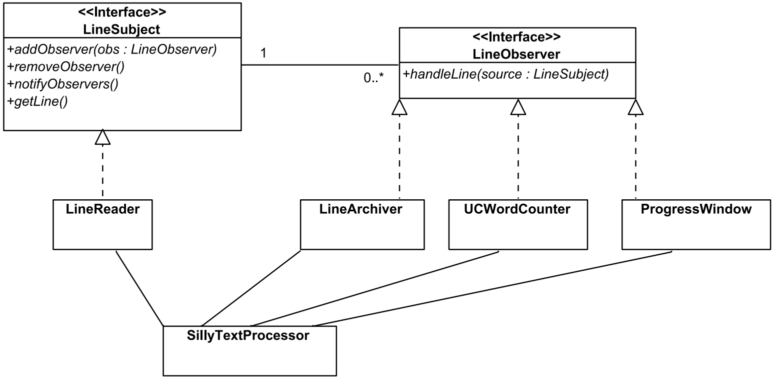


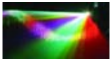
A Design that is Tightly Coupled





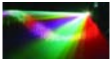
A Good Design





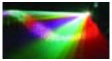
The LineObserver Interface

```
public interface LineObserver
{

    public void handleLine(LineSubject source);
}
```



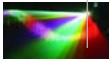
The LineSubject Interface

```
public interface LineSubject
{
    public void addObserver(LineObserver observer);


    public String getLine();



    public void notifyObservers();


    public void removeObserver(LineObserver observer);
}
```



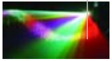
The LineReader

```
import java.io.*;
import java.util.*;

public class LineReader implements LineSubject
{
    private BufferedReader                         in;

    private List<LineObserver>                     observers;
    private int                                    maxLines;
    private String                                 line;


    public LineReader(InputStream is, int maxLines) throws IOException
    {
       this.maxLines = maxLines;

       in        = new BufferedReader(new InputStreamReader(is));
       observers = new LinkedList<LineObserver>();
    }



    public void addObserver(LineObserver observer)
    {
       observers.add(observer);
    }
```



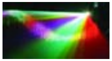
The LineReader (cont.)

```
public String getLine()
{
   return line;
}



public void notifyObservers()
{
   Iterator<LineObserver>         i;
   LineObserver                   observer;

   i = observers.iterator();
   while (i.hasNext())
   {
      observer = i.next();
      observer.handleLine(this);
   }
}


public void removeObserver(LineObserver observer)
{
   observers.remove(observer);
}
```



The LineReader (cont.)

```
    public void start() throws IOException
    {
       // Read from the console and alert listeners
       while ((line = in.readLine()) != null)
       {
          notifyObservers();
       }
    }
}
```



The SillyTextProcessor

```
// Initialization
reader   = new LineReader(System.in, maxLines);
bar      = new ProgressWindow(maxLines);
archiver = new LineArchiver();
counter  = new UCWordCounter();

reader.addObserver(bar);
reader.addObserver(archiver);
reader.addObserver(counter);

// Prompt the user
System.out.println("Enter " +maxLines +
                   " lines of text (^Z to end):\n");

reader.start();
```