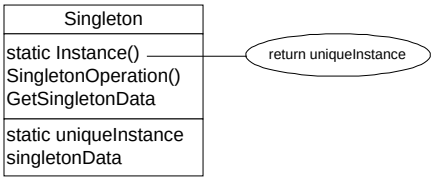
Singleton...

Structure



An Example

- I used a variation on the Singleton pattern recently when developing a data-flow analysis program. This program reads through C source code and determines all possible definitions for a given use of a variable.

```
2  int a, b;
3  a = 1;
4  b = a + 1;    use of a defined at 2
5  a = 2;
6  b = a * 2;    use of a defined at 5
```

Need to track line-number; only at one line at any given time

Example...

```
class LineInfo {
public:
    void setLineNumber (int);
   int    getLineNumber ();
};
class LastLineInfo: public LineInfo {
public:
    static LineInfo* Instance();
protected:
   LastLineInfo();
private:
    static LineInfo* _ instance;
};
```


A Common Situation

- You've partitioned your system into a collection of cooperating classes. This partitioning has the advantage that you might be able to reuse the classes. But…
  - you need to maintain consistency between related objects. To maintain reusability, you don't want to tightly couple the classes…
- Some examples...

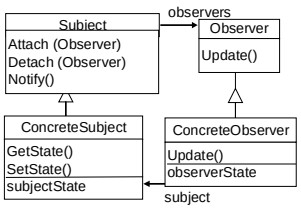
Observer Pattern...

Applicability

Use when:

- "an abstraction has two aspects, one dependent on the other. Encapsulating these aspects in separate objects lets you vary and reuse them independently.
- When a change to one object requires changing the others and you don't know how many objects need to be changed.
- When an object should be able to notify other objects without making assumptions about who those objects are." [Gang of Four, p. 294]

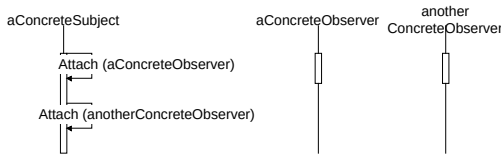
Observer Pattern: Structure



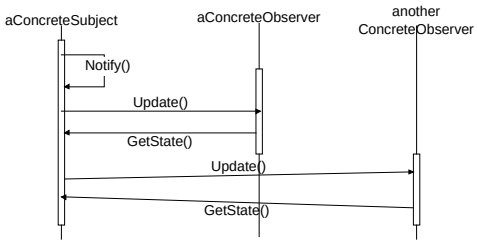
Observer Pattern: Collaborations

Collaborations

Two configurations of interest: setting up the publish-subscribe interaction; actually causing the interaction to occur.



ObserverPattern: Collaborations...



A Sample Implementation of the Observer Pattern

- In Java

```
public class Subject {
  public void attach( Observer obs ) {
    // add to list of Observers }
  public void notify() {
    // for each obs in Observers list
    //      obs->update(); }
  private /* listOfObservers
}
```

```
public class Observer {
  public void update() {
    // draw a graph
    // or something
  }
}
```

This code could be provided in a library.

A Sample Use of the Observer Pattern

```
public class Spreadsheet extends Subject {
  public void newCellValue( … ) {
    // Remember the new value
    // Then inform observers
    notify();
  }
```

```
public class Graph extends Observer {
  public void update() {
    // Better ask Subject for new
    // values and update the graph!
  }
}
```


Implementation Issues

- Observing more than one subject
  - need to extend Update interface to know which subject is notifying the Observer
- Who triggers the update?
  - State-setting operations on Subject (Observable) call Notify after they change the subject's state.
    - Advantage: Clients don't have to know about notify
    - Disadvantage: ?
      - If there are several consecutive updates it may be inefficient.

Implementation Issues...

- Who triggers the update…
  - Clients call Notify at the right time.
    - Advantage: Handle the consecutive change scenario
    - Disadvantage: Clients have to do it.
- Dangling references to deleted subjects
  - Have subject notify observers when it is destructed
- Specifying modifications of interest explicitly
  - Extend the subject's registration interface to allow observers to register for specific events

Implementation Issues...

- Avoiding observer-specific update protocols: push and pull models
  - Push: subject sends observers detailed information about change whether they want it or not
  - Pull: Subject sends nothing; observers ask for details explicitly
  - The push model may make observers less reusable. The pull model may be inefficient

Other Uses of ObserverPattern

- Smalltalk Model/View/Controller (MVC)
- Smalltalk and ET++ provide a general dependency mechanism of Subject/Observer in the parent class of all other classes in the system
- InterViews, Andrew, Unidraw

Some Patterns...

- Abstract Factory
- Builder
- Factory Method
- Prototype
- Singleton

Creational

- Adapter
- Bridge
- Composite
- Decorator

Structural

- Façade
- Flyweight
- Proxy

- Chain of Responsibility
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer...

Behavioural