Observer

Design Patterns

Outline

- Motivating Example
- Introduction to the Observer

- Traditional
- Events and Delegates
- IObservable<T>

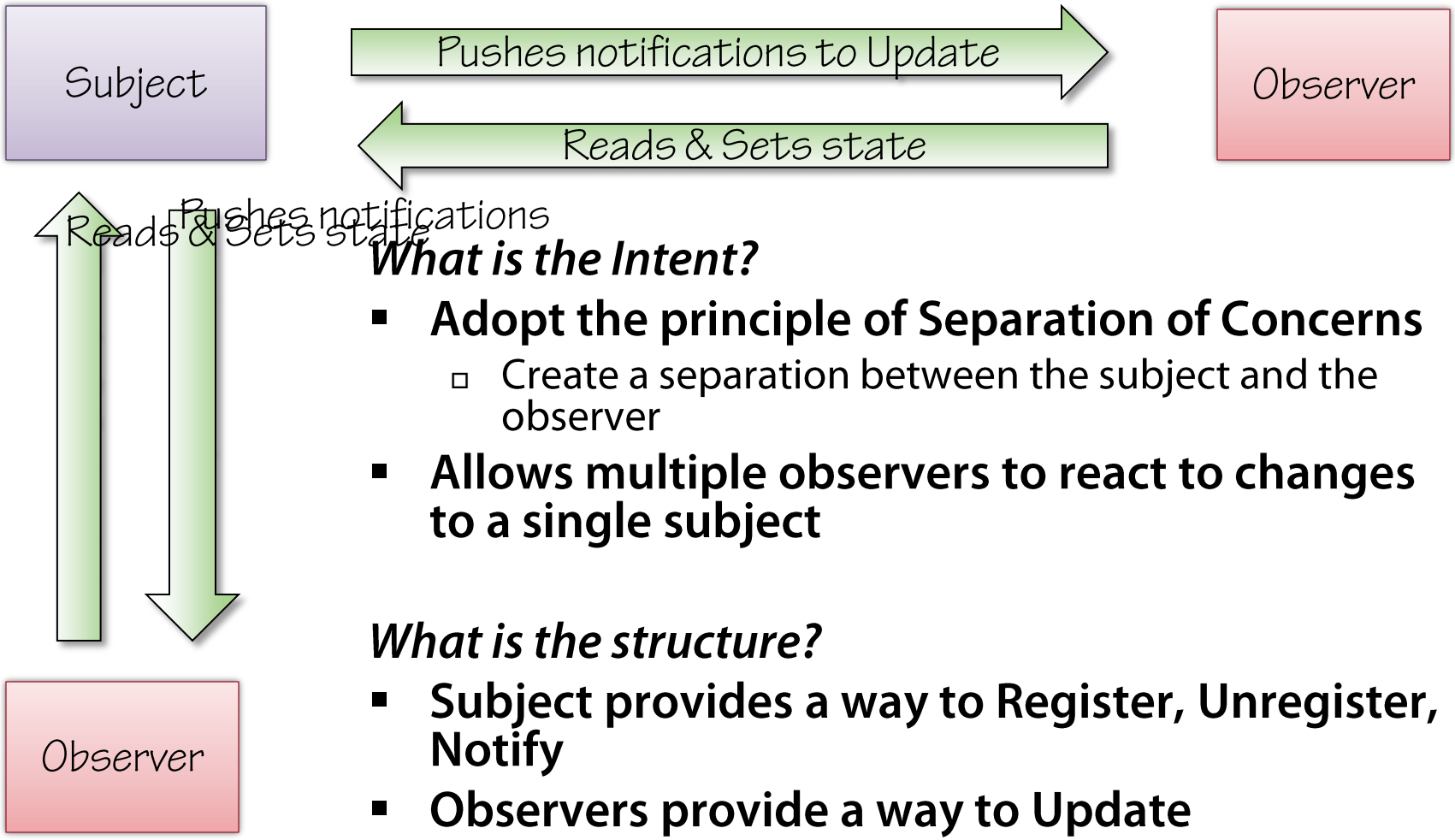
- Real World Examples
- Pitfalls to Avoid

Motivating Example

Consider Observers when:

- When one object is dependent on another object.
- When changing one object requires a change to many others
- When changes to an object should allow notification to others without any knowledge of them

Introduction to the Observer



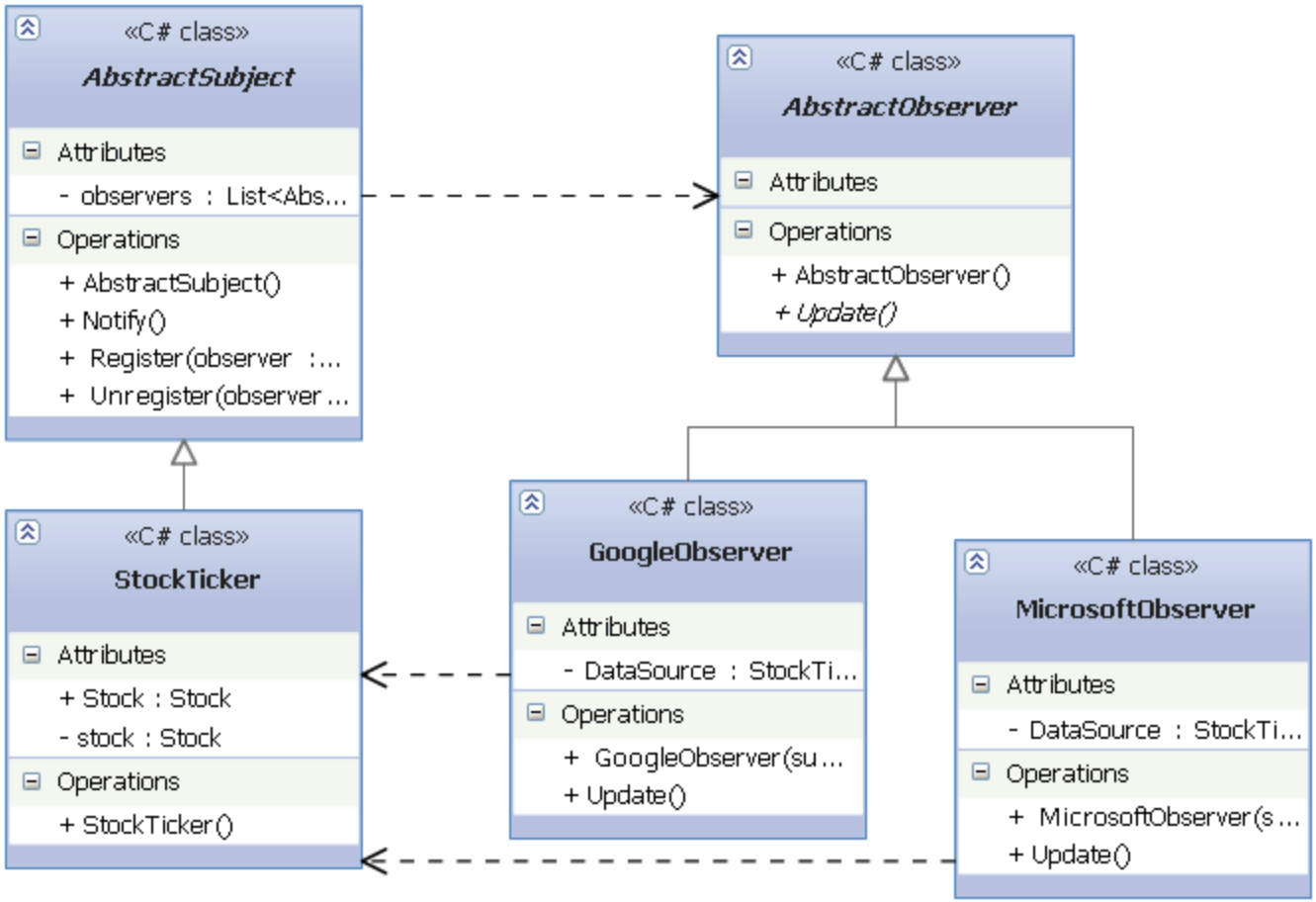
What is the Intent?

- **Adopt the principle of Separation of Concerns**
  - Create a separation between the subject and the observer
- **Allows multiple observers to react to changes to a single subject**

What is the structure?

- **Subject provides a way to Register, Unregister, Notify**
- **Observers provide a way to Update**

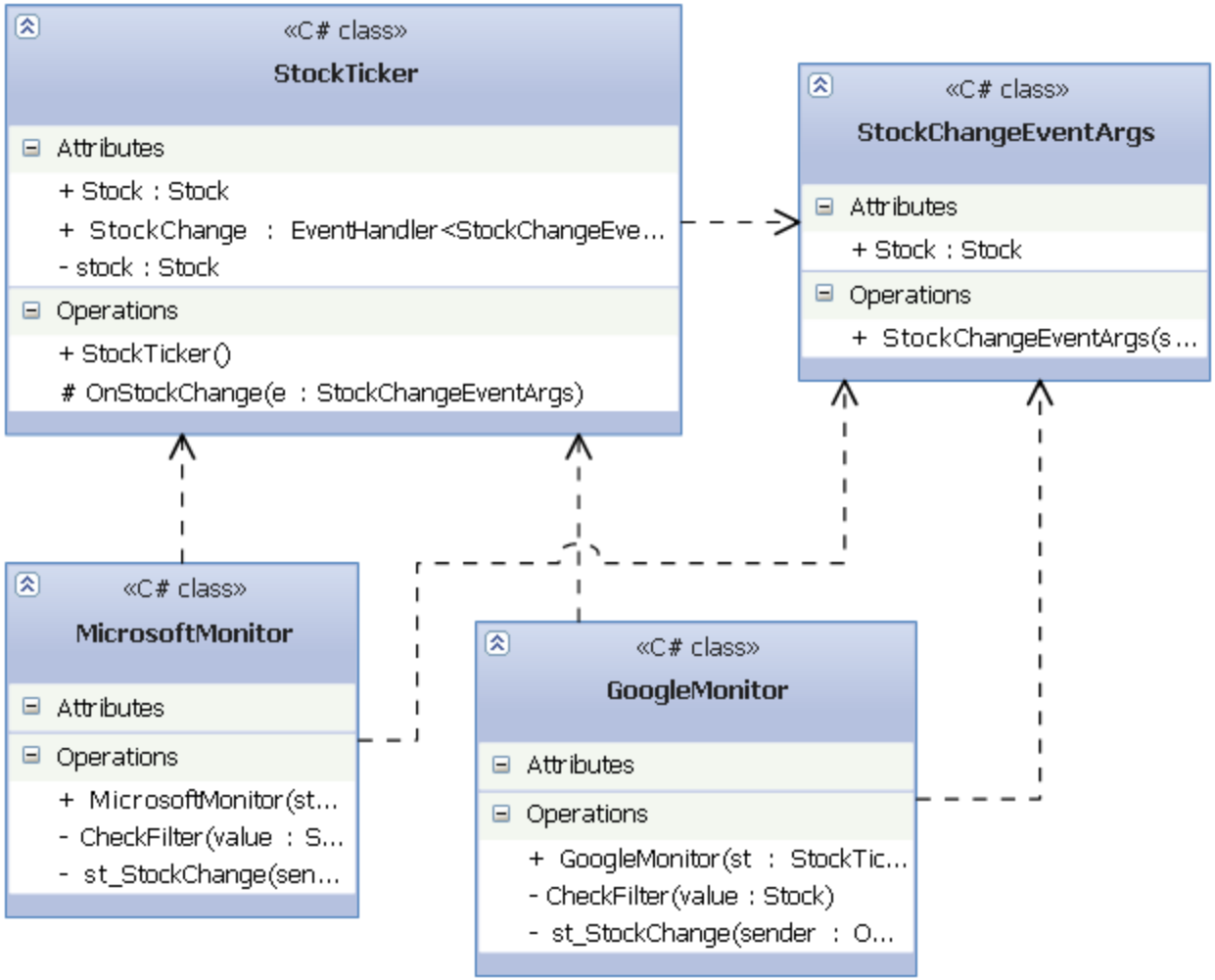
Traditional Approach



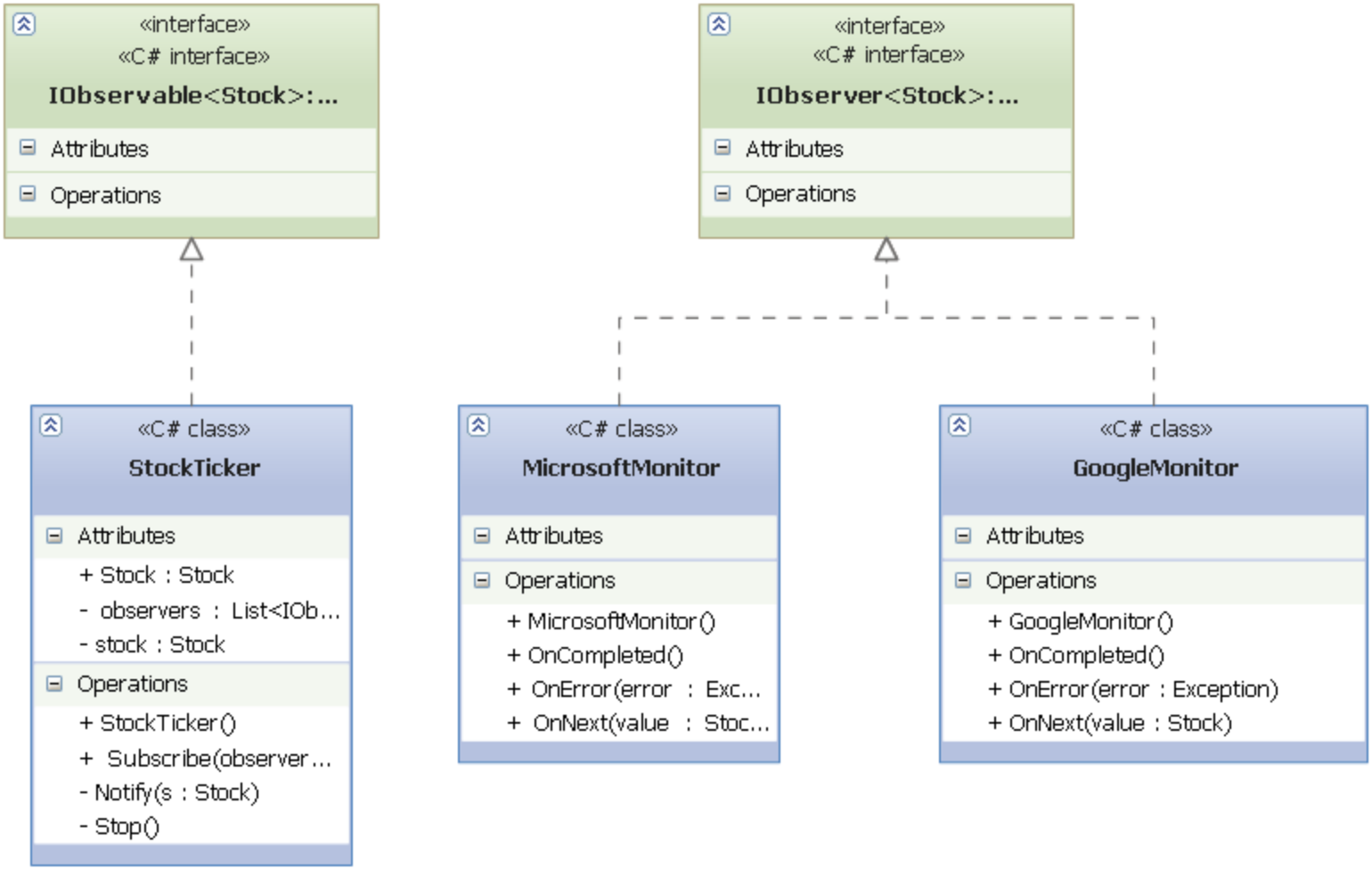
Consequences

- Multiple subjects for each observer
- Triggering the update, when multiple properties are involved
- Disposed subjects & observers can hold references to each other
- Mapping of subjects to their observers
- Unexpected updates
- Observing different properties separately

Events and Delegates



IObserver<T>


«interface»
«C# interface»
IObservable<Stock>:...
Attributes
Operations
«interface»
«C# interface»
IObserver<Stock>:...
Attributes
Operations
«C# class»
StockTicker
Attributes
+ Stock : Stock
- observers : List<IOb...
- stock : Stock
Operations
+ StockTicker()
+ Subscribe(observer...
- Notify(s : Stock)
- Stop()
«C# class»
MicrosoftMonitor
Attributes
Operations
+ MicrosoftMonitor()
+ OnCompleted()
+ OnError(error : Exc...
+ OnNext(value : Stoc...
«C# class»
GoogleMonitor
Attributes
Operations
+ GoogleMonitor()
+ OnCompleted()
+ OnError(error : Exception)
+ OnNext(value : Stock)

Push – versus -Pull


```
interface IEnumerable<out T>
{
   IEnumerator<T> GetEnumerator();
}

interface IEnumerator<out T>: IDisposable
{
   bool MoveNext();   // throws Exception
   T Current { get; }
}
```

```
interface IObservable<out T>
{
  IDisposable Subscribe(IObserver<T> observer);
}

interface IObserver<in T>
{
  void OnCompleted(bool done);
  void OnError(Exception exception);
  T OnNext { set; }
}
```

- Provides a Pull based interface
- Dual of IObservable

- Provides a Push based interface
- Dual of IEnumerable

Real World Examples & Pitfalls

Real World Examples

- GUI Controls
- Data Binding
- Network Events
- File Watcher
- Model View Controller pattern

Pitfalls to Avoid

- Unexpected Thread
- Multiple Threads
- Memory Leaks

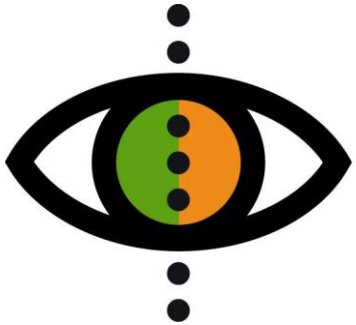
Summary

- Stock Ticker
- Separation of Concerns

- Traditional
- Events and Delegates
- IObservable<T>

- File Watchers & Network Change
- Pitfalls to Avoid

For more in-depth online developer training visit



pluralsight

see what you can learn

on-demand content from authors you trust

References

- **Design Patterns, Elements of Reusable Object-Oriented Software**
  1995, Gamma, Helm, Johnson, Vlissides

- **Subject/Observer is Dual to Iterator**
  http://csl.stanford.edu/~christos/pldi2010.fit/meijer.duality.pdf