

GoF Observer



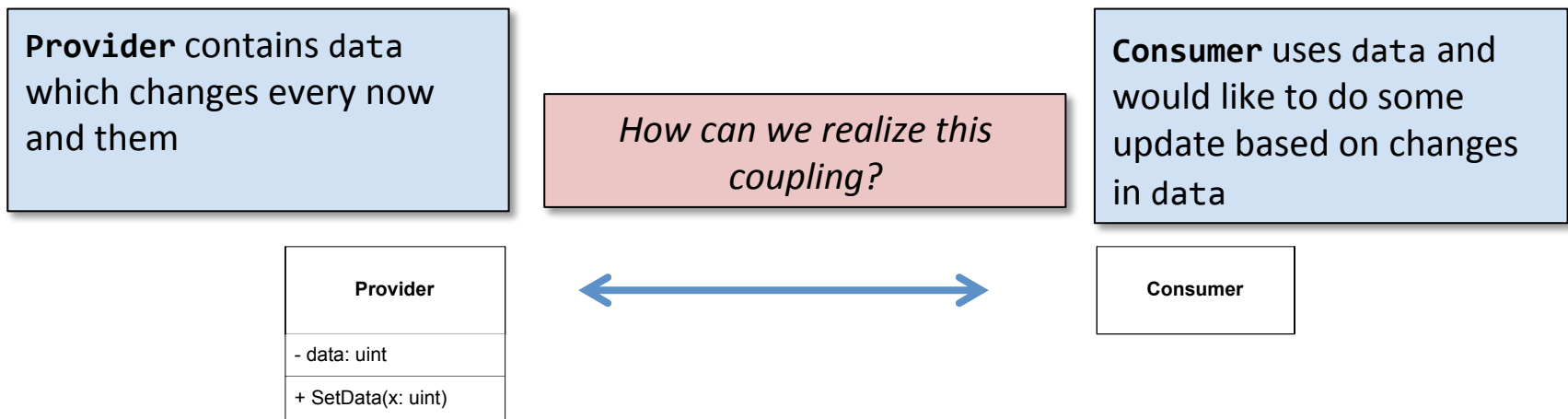
Image source: www.dofactory.com
Slides modified from original slides by Troels Fedder Jensen

Outline

- GoF Observer
- Introduction to lab exercise
- Creating groups
- Lab

Decoupling the general “data-update” problem

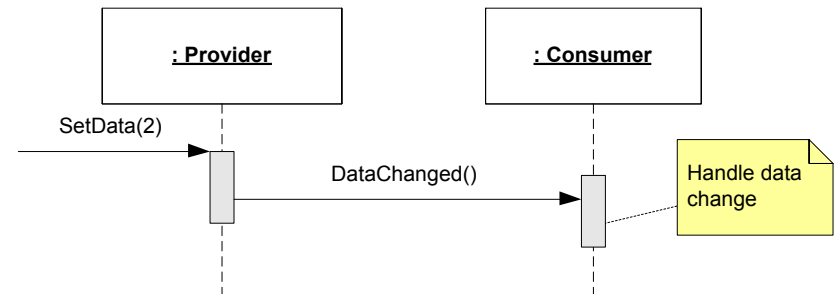
- Let’s take a look at the general “data-update” problem.



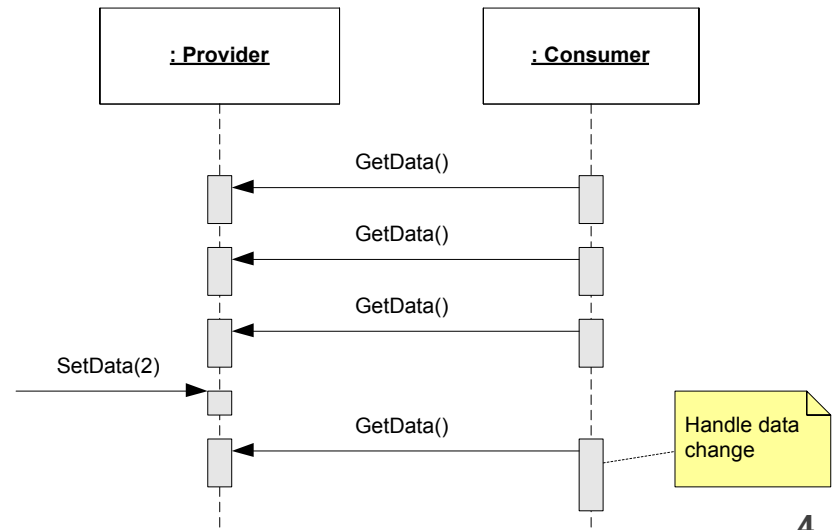
Decoupling the general “data-update” problem

- Discuss pros and cons of these typical solutions

- Provider notifies Consumer when data changes



- Consumer polls Provider



GoF Observer to the rescue!

- We need some mechanism that...
 - allows Consumers to be added to the Provider without changing the Provider (i.e. adhere to OCP)
 - allows Provider to inform Consumers of data changes (i.e. promotes low coupling)
 - allows many Consumers to be informed on updates of same data
- We need....GoF Observer!

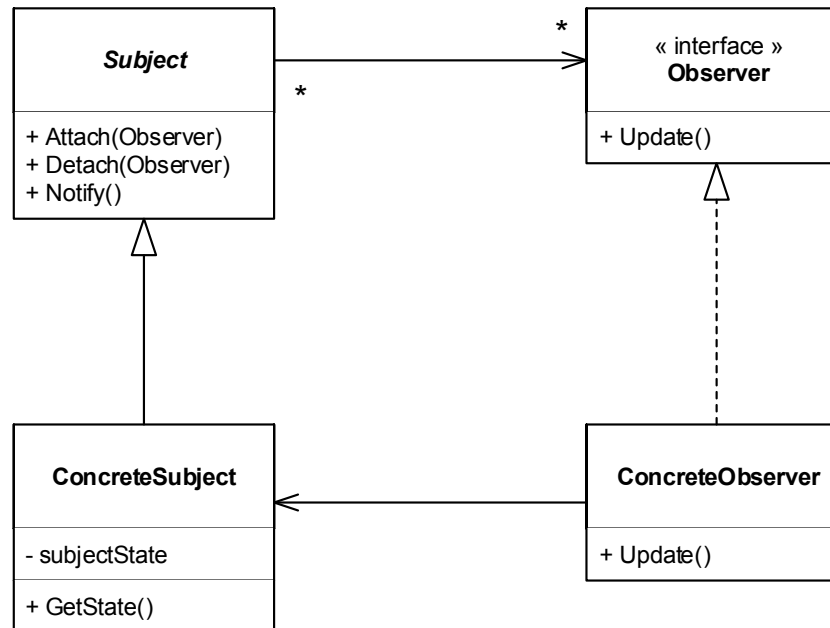
Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically



GoF Observer

Subject is an abstract base class for all data subjects (i.e. things that get *updated*)

Observer is an interface that must be implemented by all classes that wish to be informed of data changes

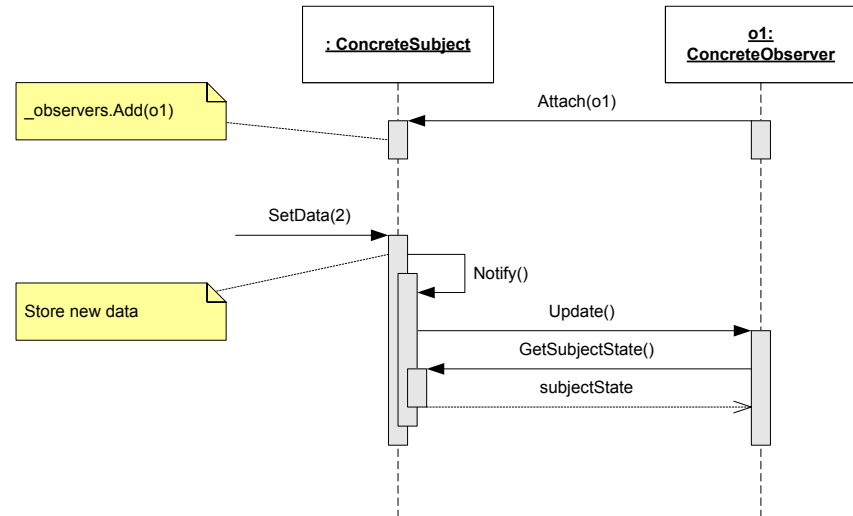


ConcreteSubject inherits from Subject. This is the actual class that must be monitored (in our case, Provider)

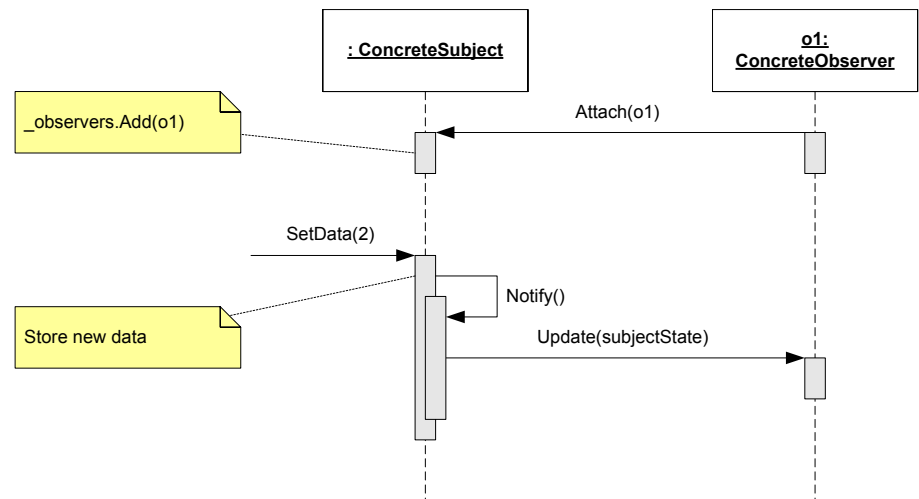
ConcreteObserver implements the Observer interface. This is the actual class that must receive updates (in our case, Po)

GoF Observer – pull and push variants

- *Pull* variant
(Observer *pulls* state from subject)

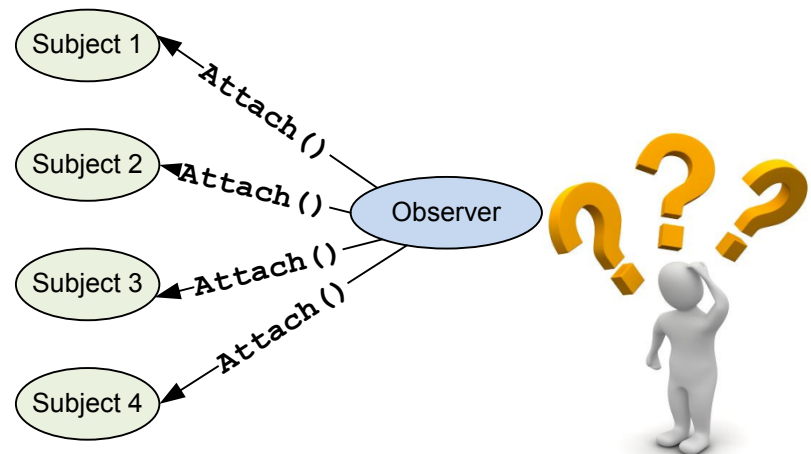
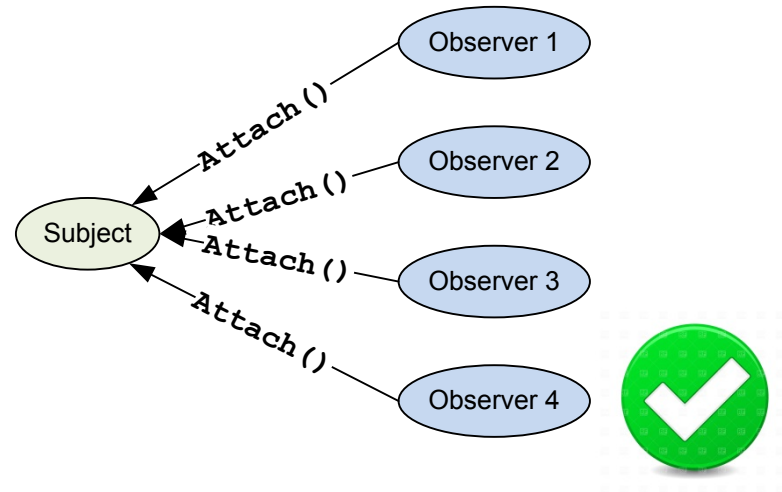


- *Push* variant
(Subject *pushes* state to observer)



GoF Observer – handling several subjects of same type

- The variant of GoF Observer we have studied handles several observers registering on the *same* subject
- How about one Observer attaching to several subjects of the same type?



Handling several subjects of same type


- Subject sends reference-to-self

```
class SomeSubject : Subject
{
    public void SetState(State state)
    {
        _state = state;
        NotifyObservers(this);
    }
}
```

```
class SomeObserver : Observer
{
    public void AddSubject(Subject s)
    {
        s.Attach(this);
    }

    public void Update(Subject s)
    {
        // Do something with 's'
    }
}
```

Sending this uniquely
ID's the Subject



Handling several subjects of same type

– Subject sends tag

```
class SomeSubject : Subject
{
    string tag;

    public void SetState(State state)
    {
        _state = state;
        NotifyObservers(tag);
    }
}
```

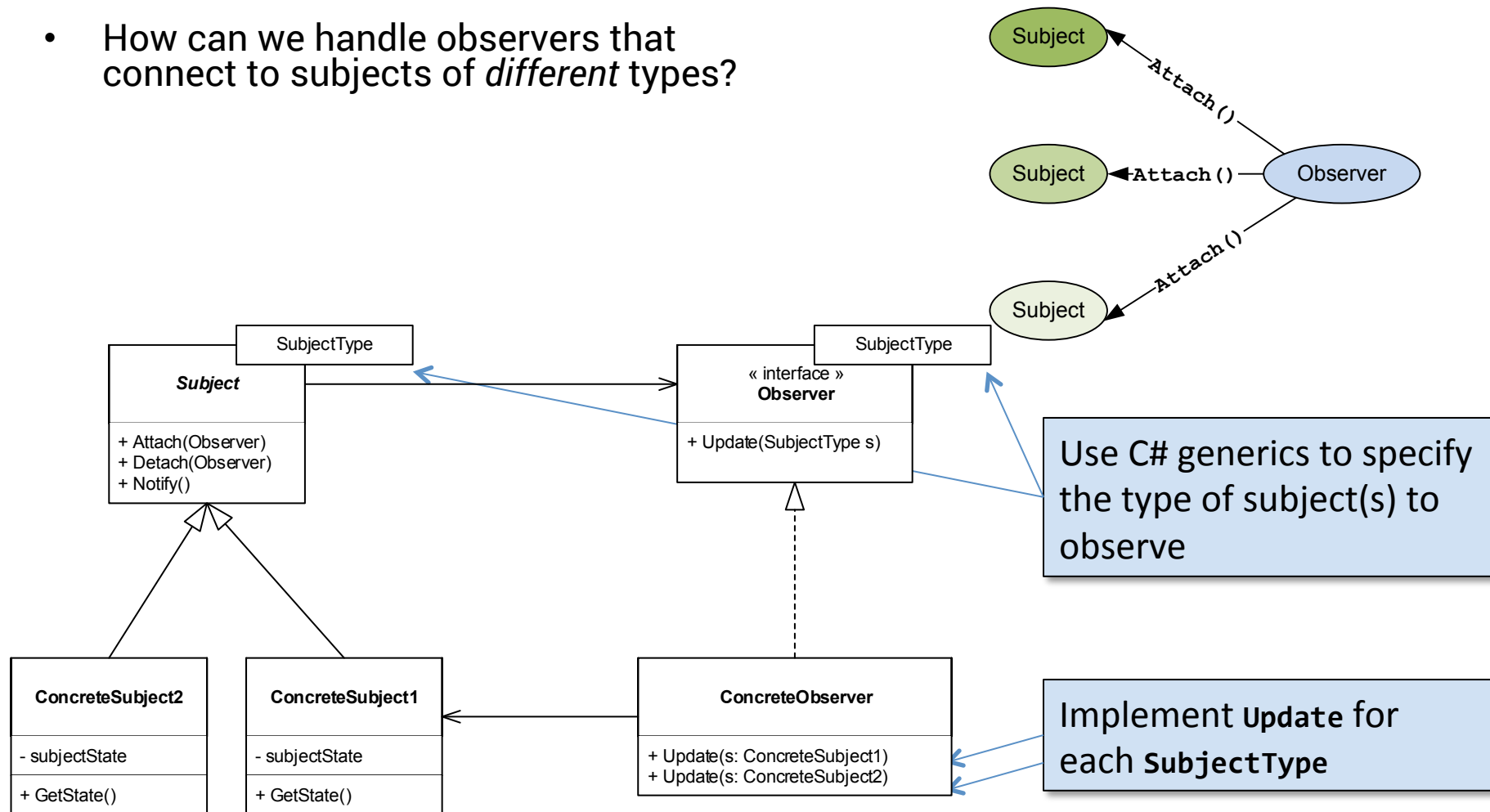
```
class SomeObserver : Observer
{
    public void AddSubject(Subject s)
    {
        s.Attach(this);
    }

    public void Update(string tag)
    {
        // Do something with the Subject
        // ID'ed by 'tag'
    }
}
```

Sending tag also ID's the Subject, but does not send an object reference. It is up to the observer to find the correct reference

GoF Observer – handling subjects of different types

- How can we handle observers that connect to subjects of *different* types?



GoF Observer – handling subjects of different types

```
public class CoordinateSubject : Subject<CoordinateSubject>
{
    public CoordinateSubject()
    {
        X = 0;
        Y = 0;
    }

    public double X { get; private set; }
    public double Y { get; private set; }

    public void SetX(double x)
    {
        if (Math.Abs(X - x) > Double.Epsilon)
        {
            X = x;
            NotifyObservers();
        }
    }
    ...
}
```

```
public class MyComplexObserver
    : IObservable<CoordinateSubject>
    IObservable<SomeOtherSubject>
{
    public void Update(CoordinateSubject subject)
    {
        // Do something with CoordinateSubject
    }

    public void Update(SomeOtherSubject subject)
    {
        // Do something with SomeOtherSubject
    }
}
```

