Patrons de conception 02

Objectifs

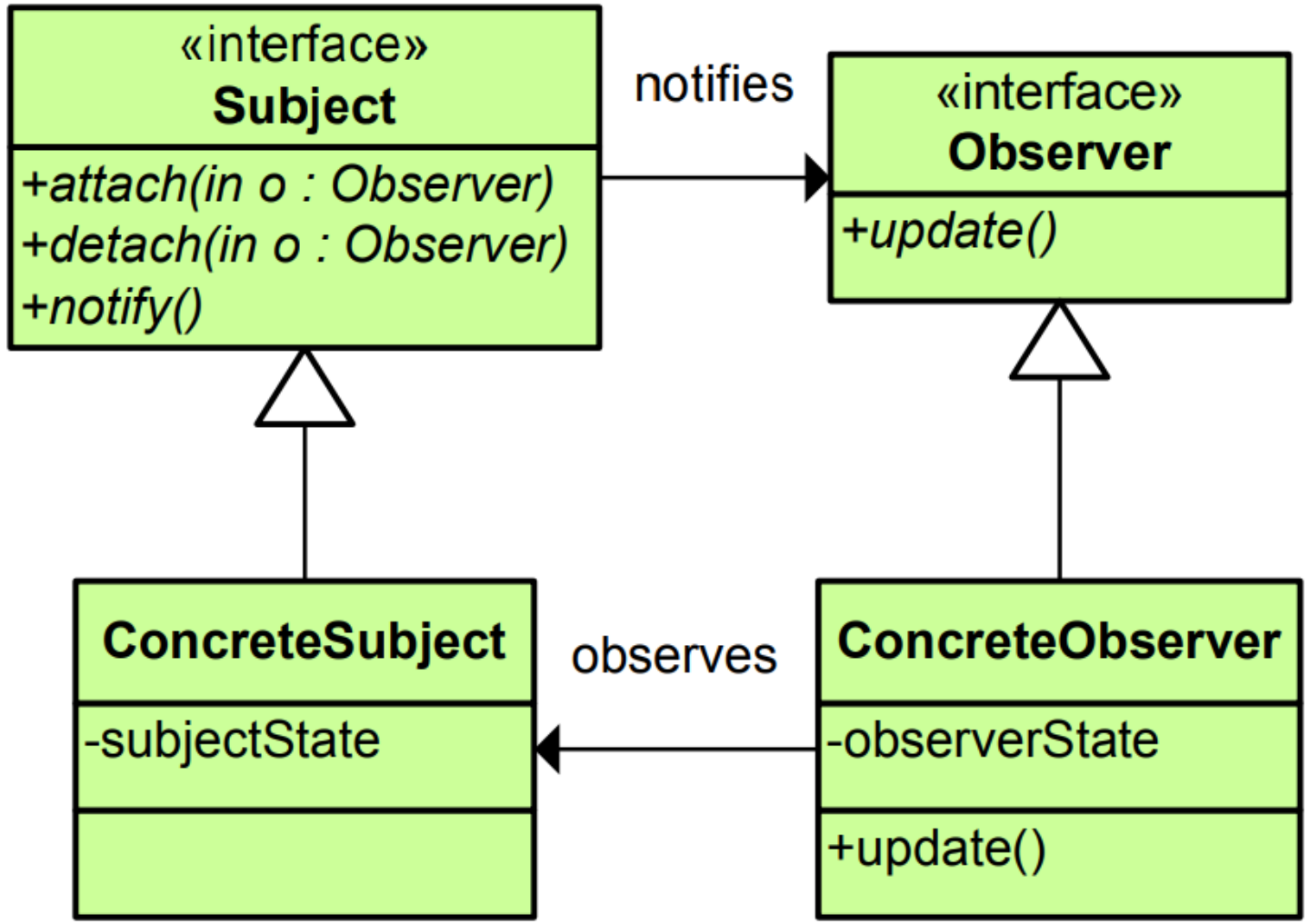
- Comprendre le patron de conception « Observateur »
- Implanter ce patron en C#

Observateur

Observer

Type: Behavioral

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



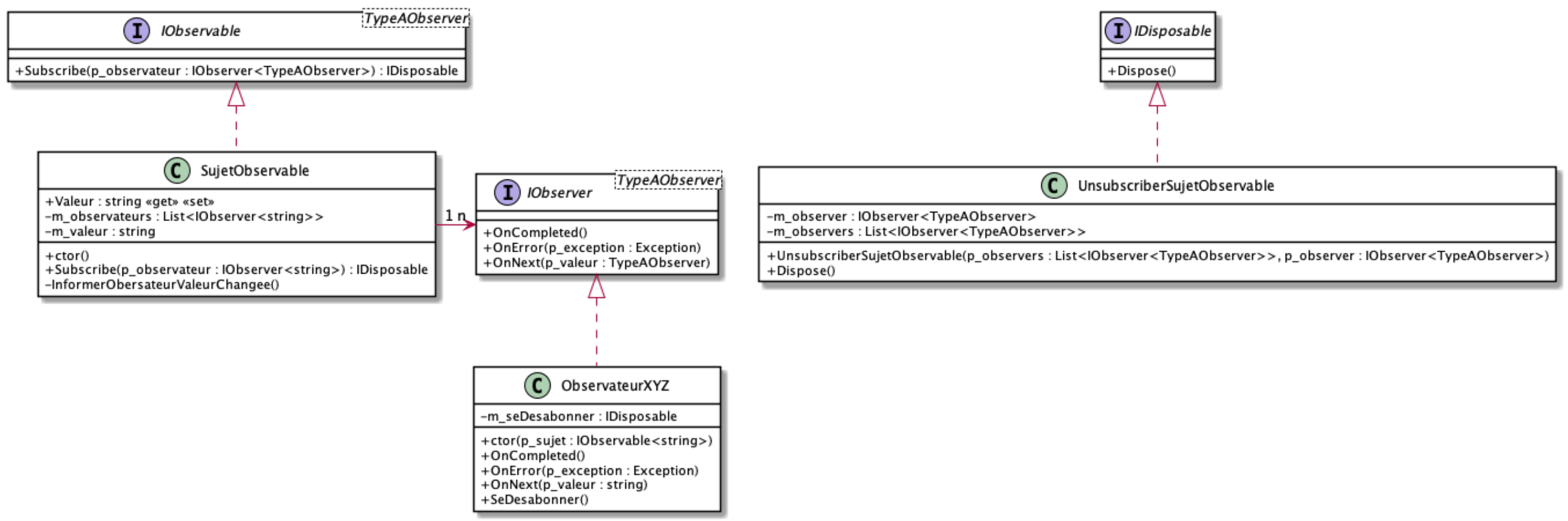
Observateur et synonymes

- Le **sujet** à observer peut être appelé un **observable** ou un **fournisseur**
- L'**observateur** peut être appelé un **consommateur**
- Quand un **observateur veut observer un objet observable**, on dit qu'il s'attache ou qu'il s'abonne

Implantation C#

- Dans le cadriciel .Net core, ce patron peut-être implanté au travers des interfaces génériques :
  - **IObservable<Type>** (sujet) : https://docs.microsoft.com/en-us/dotnet/api/system.iobservable-1
    - **IDisposable Subscribe(IObserver<T> observer)** : ajout un observateur, et renvoie un objet qui permet de d'enlever l'observateur
  - **IObserver<Type>** (observateur) : https://docs.microsoft.com/en-us/dotnet/api/system.iobserver-1
    - **void OnCompleted()** : appelée dès que le sujet a terminé son travail et / ou sa durée de vie
    - **void OnError(Exception error)** : appelée dès qu'une exception est détectée, ne doit pas lever d'exception
    - **void OnNext(Type value)** : appelée dès qu'il y a un changement

Diagramme de l’exemple



Exemple implantation C# - Sujet

```
public class SujetObservable : IObservable<string> {
    private List<IObserver< string >> m_observateurs;
    private string m_valeur;

    public SujetObservable() {
        this.m_observateurs = new List<IObserver< string >>();
    }

    public IDisposable Subscribe(IObserver< string > p_observateur) { // PFL : attach
        if (p_observateur is null) {
            throw new ArgumentNullException(nameof(p_observateur));
        }

        this.m_observateurs.Add(p_observateur);

        return new UnsubscribeSujetObservable(this.m_observateurs, p_observateur);
    }

    public string Valeur {
        get { return this.m_valeur; }
        set {
            this.m_valeur = value;
            this.InformerObersateurValeurChangee();
        }
    }

    private void InformerObersateurValeurChangee() { // PFL : notify
        this.m_observateurs.ForEach(observer => observer.OnNext(this.Valeur) );
    }
}
```

Exemple implantation C# - Observateur

```
public class ObservateurXYZ : IObserver< string > {
    private IDisposable m_seDesabonner;

    public ObservateurXYZ(IObservable<string> p_sujet) {
        if (p_sujet is null) {
            throw new ArgumentNullException(nameof(p_sujet));
        }

        this.m_seDesabonner = p_sujet.Subscribe(this);
    }

    public void OnCompleted() {
        // PFL : autre code
        this.SeDesabonner();
    }

    public void OnError(Exception error) {
        // PFL : autre code
    }

    public void OnNext(string p_valeur) {  // update
         // PFL : autre code
    }

    public void SeDesabonner() {
        this.m_seDesabonner?.Dispose();
        this.m_seDesabonner = null;
    }
}
```

Exemple implantation C# - Observateur - désabonnement

```
public class UnsubscriberSujetObservable : IDisposable
{
    private IObserver< string > m_observateur;
    private List<IObserver< string >> m_observateurs;

    public UnsubscriberSujetObservable(List<IObserver< string >> p_observateurs, IObserver< string > p_observateur)
    {
        this.m_observateur = p_observateur;
        this.m_observateurs = p_observateurs;
    }

    // PFL : dettach
    public void Dispose()
    {
        this.m_observateurs.Remove(this.m_observateur);
    }
}
```

Démo

- Voir code sur le dépot Git