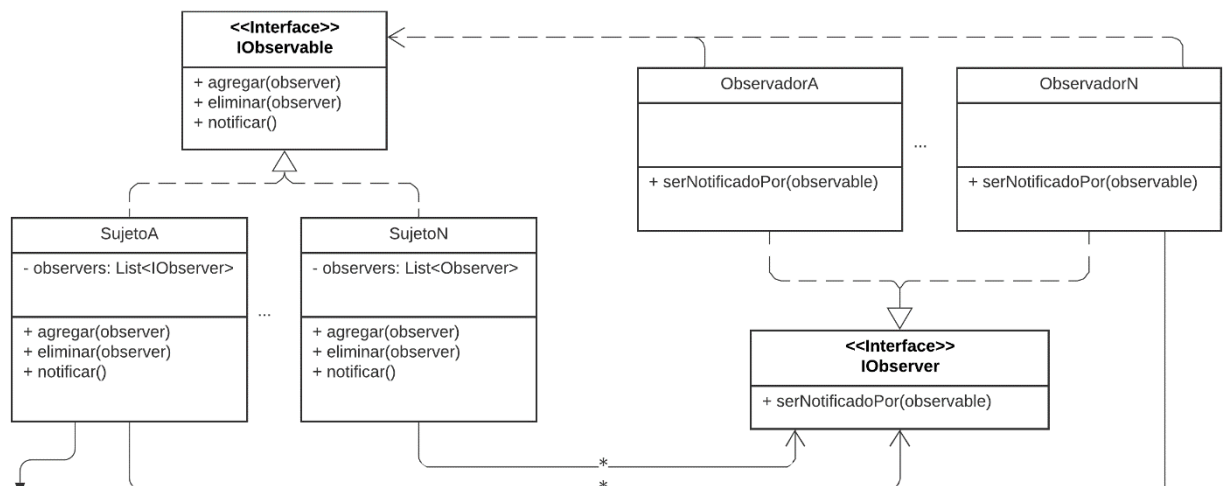


Patrón Observer

- **Clasificación:** Patrón de comportamiento
- **¿Qué hace?**
 - ✓ Permite notificar a los interesados de cierto sujeto los eventos que a éste le ocurren y que ellos actúen en consecuencia.
 - ✓ Desacopla las acciones que se ejecutan de los eventos que las activan.
- **Se sugiere su utilización cuando:**
 - ✓ Se requiere disparar acciones de diferentes naturalezas frente a la ocurrencia de un evento en algún objeto.
 - ✓ Se requiere notificar/avisar a algunos interesados de un objeto sobre la ocurrencia de algún evento y no interesa saber quiénes son y/o cuántos son.
- **Estructura genérica:**



```

>> public class SujetoA implements IObservable {
    private List<IObservable> observers;

    public SujetoA(){
        this.observers = new ArrayList<IObservable>();
    }

    public void agregar(IObservable observer){
        this.observers.add(observer);
    }

    public void eliminar(IObservable observer){
        this.observers.remove(observer);
    }

    public void notificar(){
        this.observers.forEach(o -> o.serNotificadoPor(this));
    }
}
    
```

```

>> public class ObservadorN implements IOObserver {

    public void serNotificadoPor(IObservable observable) {
        // hacer algo
    }
}
    
```

Consideraciones:

- ✓ Las interfaces (IObservable, IObserver) pueden ser reemplazadas por clases abstractas que definan el comportamiento en común.
 - ✓ Pueden no existir N observers concretos.
 - ✓ Pueden no existir N sujetos concretos.
 - ✓ El método *notificar* del sujeto es de ejecución sincrónica. Si se requiere que la ejecución sea asincrónica, se deberán realizar las modificaciones correspondientes para cada lenguaje de programación.
 - ✓ El método *notificar* del sujeto es de ejecución secuencial. Si se requiere que la ejecución sea paralela, se deberán realizar las modificaciones correspondientes para cada lenguaje de programación.
- **¿Qué proporciona su uso?**
 - ✓ Mayor cohesión a la clase Sujeto.
 - ✓ Mayor mantenibilidad debido a que el comportamiento establecido por cada observador es fácilmente localizable.
 - ✓ Extensibilidad a causa del desacoplamiento entre acción - reacción.
 - ✓ Mayor facilidad para reutilización de los observadores.
 - **Code smells que soluciona/evita de forma directa:**
 - ✓ Métodos largos
 - ✓ Código duplicado
 - ✓ Clase Dios