

Делегати і події як елементи ООП в С#

Члени класу

- **Поля (fields)** – змінні об'єкта:
 - зберігають дані про стан об'єкта
 - спосіб використання полів визначають *static*, *readonly* і *const*
- **Методи (methods)** – код, який задає дії над об'єктами та їх даними
- **Константи (constants)** – поля, значення яких є однаковими в кожного об'єкта, відомі на момент компілювання і не можуть змінюватися за весь час існування об'єкта
- **Властивості (properties)** – методи опосередкованого доступу до даних об'єкта:
 - реалізація аксесорними методами *get* і *set*
 - з точки зору клієнта класу виглядають як поля
- **Індексатори (indexers)** – засоби доступу до полів об'єкта за значенням індексу; реалізація аксесорними методами *get* і *set*
- **Делегати (delegates)** – типи, спеціалізовані *delegate* – засоби опосередкованого виклику методів:
 - екземпляр делегата інкапсулює список виклику з одним або кількома є методів, доступних для виклику
 - на момент компілювання має бути заданою лише сигнатура методів
 - конкретні реалізації методів призначаються в процесі виконання програми
 - делегати як типи можуть бути зовнішні, так і вкладені в якийсь клас
- **Події (events)** – спеціалізовані *event* делегати; разом з оголошенням свого типу-делегата вони публікуються (*public*-специфіковані) класом, об'єкти якого при виконанні певних умов через делегати-події викликатимуть потрібний метод (обробітник події) іншого класу, який в своїх методах підписався на конкретну подію.
- **Оператори (operators)** – перевантажені для класу стандартні оператори, які дають змогу використовувати об'єкти класу подібно до об'єктів вбудованих типів.
- **Вкладені (nested) типи** – для створення об'єктів лише для внутрішнього використання

Делегати

атрибути_{opt} модифікатори_{opt}

delegate тип результату ідентифікатор (список аргументів)

❑ Делегат – спеціалізований тип-посилання

- забезпечує механізм пізнього зв'язування
- вказує сигнатуру методів, які можуть бути долучені до делегата

❑ Об'єкт делегата може долучити у список виклику:

- довільний іменований метод з відповідною сигнатурою(**variance support**)
- анонімний метод

Design Goals for Delegates

Delegates provide a **late binding** mechanism in .NET:

- a common language construct that could be used for any late binding algorithms
- to support both **single** and **multi-cast** method calls
- to support the same type safety that developers expect from all C# constructs
- to ensure that the code for delegates could provide the basis for **event** pattern

Об'єкт-делегат

- ❑ Назва **типу** делегата визначена його назвою

```
public delegate void inform(Point p, Point q);
```

- ❑ Змінна-делегат – ініціалізується і може бути використана як звичайний **об'єкт**:

- інстанціювання об'єкта-делегата

```
inform pi = pn.print_dist;
```

- зберігає стан (значення)
- може бути використана у різних структурах даних
- передають як аргумент у методи ...

- ❑ Змінна-делегат має значення **null** , якщо список викликів порожній:

- при спробі виклику через делегат – виняток **NullReferenceException**
- спроба вилучення зі списку виклику незареєстрованого методу не змінює стан делегата, жоден виняток не генерується

Виклик методів через делегат

Алгоритм виклику через об'єкт-делегат методів, які зареєстровані у його списку викликів:

1. передача даних, отриманих делегатом через аргументи, відповідним аргументам усіх зареєстрованих методів
 - якщо аргументом є об'єкт-посилання, то усі методи отримують посилання на той самий об'єкт
 - зміна об'єкта, переданого через аргумент-посилання, одним методом зі списку викликів будуть видимими для методів, що знаходяться далі у списку викликів
2. виклик методу, виконання його інструкцій
3. повернення результату виконання методу (якщо передбачено) в делегат
4. повернення отриманого в методі результату в місце виклику делегата

Делегат як тип

- ❑ неявно оголошений як **sealed** (не допускає утворення похідних класів)
- ❑ може бути компонентою типів-агрегатів (вкладений делегат)
- ❑ ініціалізують як статичними (через тип), так і нестатичними методами
- ❑ об'єднання делегатів
- ❑ використання з подіями

Ініціалізація іменованими методами

```
delegate void Del();

class SampleClass
{
    public void InstanceMethod() { ....}
    static public void StaticMethod() {.....}
}

class TestSampleClass
{
    static void Main()
    {
        SampleClass sc = new SampleClass();

        Del d = sc.InstanceMethod;
        d();

        d = SampleClass.StaticMethod;
        d();
    }
}
```


Ініціалізація неіменованими методами

```
delegate void Del(int x);  
// Instantiate the delegate using an anonymous method  
Del d = delegate (int k) { /* ... */ };
```

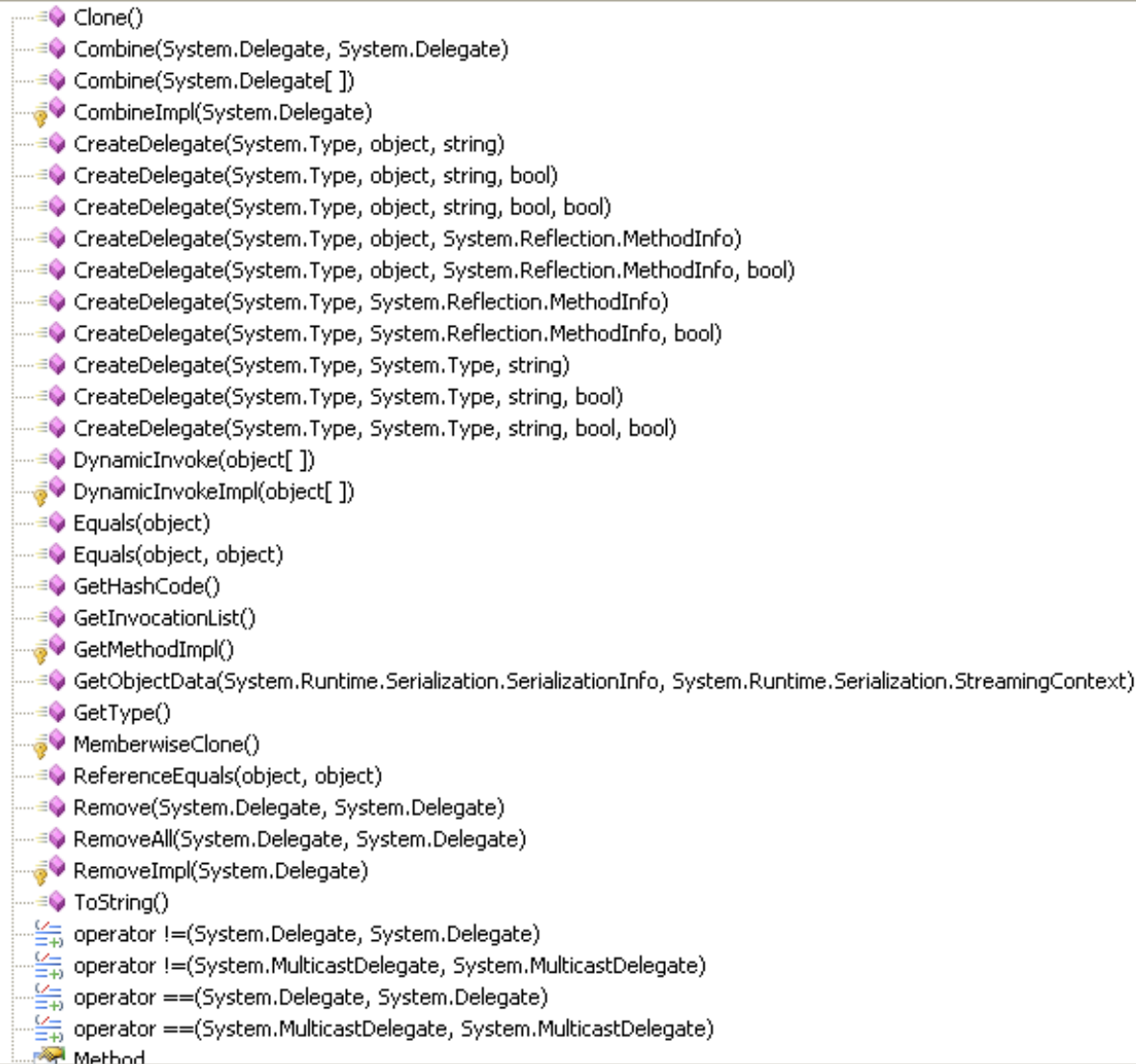
```
// using an anonymous method without arguments  
test(delegate{ Console.WriteLine("anonymous_method");}, parr);
```

Композитні делегати

Multicast Delegates

- ❑ Керування списком методів операторами
 - `+`, `+=` додавання до списку нових методів
 - `-`, `-=` вилучення методу зі списку
- ❑ Тип результату кожного з методів `void`
- ❑ Жоден з параметрів - не `out`

public delegate MulticastDelegate



A screenshot of the Visual Studio IDE showing the `MulticastDelegate` class. The class is a public delegate and a member of the `System` namespace. The methods listed are:

- `Clone()`
- `Combine(System.Delegate, System.Delegate)`
- `Combine(System.Delegate[])`
- `CombineImpl(System.Delegate)`
- `CreateDelegate(System.Type, object, string)`
- `CreateDelegate(System.Type, object, string, bool)`
- `CreateDelegate(System.Type, object, string, bool, bool)`
- `CreateDelegate(System.Type, object, System.Reflection.MethodInfo)`
- `CreateDelegate(System.Type, object, System.Reflection.MethodInfo, bool)`
- `CreateDelegate(System.Type, System.Reflection.MethodInfo)`
- `CreateDelegate(System.Type, System.Reflection.MethodInfo, bool)`
- `CreateDelegate(System.Type, System.Type, string)`
- `CreateDelegate(System.Type, System.Type, string, bool)`
- `CreateDelegate(System.Type, System.Type, string, bool, bool)`
- `DynamicInvoke(object[])`
- `DynamicInvokeImpl(object[])`
- `Equals(object)`
- `Equals(object, object)`
- `GetHashCode()`
- `GetInvocationList()`
- `GetMethodImpl()`
- `GetObjectData(System.Runtime.Serialization.SerializationInfo, System.Runtime.Serialization.StreamingContext)`
- `GetType()`
- `MemberwiseClone()`
- `ReferenceEquals(object, object)`
- `Remove(System.Delegate, System.Delegate)`
- `RemoveAll(System.Delegate, System.Delegate)`
- `RemoveImpl(System.Delegate)`
- `ToString()`
- `operator !=(System.Delegate, System.Delegate)`
- `operator !=(System.MulticastDelegate, System.MulticastDelegate)`
- `operator ==(System.Delegate, System.Delegate)`
- `operator ==(System.MulticastDelegate, System.MulticastDelegate)`

Method

public delegate **MulticastDelegate**
Member of [System](#)

Summary:

Represents a multicast delegate; that is, a delegate that can have more than one element in its invocation list.

Делегати-узагальнення

Тип делегата може мати параметри

```
public delegate void D<T>(T item);  
  
public static void Notify(int i) {...}  
  
var m1 = new D<int>(Notify);  
D<int> m2 = Notify;
```

Strongly typed system delegates

- ❑ Variations of **Action** delegate with up to 16 arguments such as

Action<T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16>

```
public delegate void Action();  
public delegate void Action<in T>(T arg);  
public delegate void Action<in T1, in T2>(T1 arg1, T2 arg2);  
. . . . .
```

- ❑ Variations of **Func** delegate with up to 16 input arguments such as

Func<T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16, TResult>

The type of the result is always the last type parameter in all the Func declarations

```
public delegate TResult Func<out TResult>();  
public delegate TResult Func<in T1, out TResult>(T1 arg);  
public delegate TResult Func<in T1, in T2, out TResult>(T1 arg1, T2 arg2);  
. . . . .
```

- ❑ Specialized **Predicate<T>** type for a delegate that returns a test on a single value

```
public delegate bool Predicate<in T>(T obj);
```

Приклад використання стандартного делегата

```
public static class Logger
{
    public static Action<string> WriteMessage;

    public static void LogMessage(string msg)
    {
        WriteMessage(msg);
    }
}
```

```
public static void LogToConsole(string message)
{
    Console.Error.WriteLine(message);
}
```

```
. . .
Logger.WriteMessage += LogToConsole;
. . .
```

Вкладені делегати

- ❑ Вкладений в узагальнення делегат може використовувати параметри цього узагальнення

```
class Stack<T>
{
    T[] items;
    public delegate void StackDelegate(T[] tms);
}
```

- ❑ Специфікація параметра узагальнення

```
private static void DoWork(float[] items) { }
public static void TestStack()
{
    Stack<float> s = new Stack<float>();
    Stack<float>.StackDelegate d = DoWork;
}
```

Події

атрибути_{opt}

модифікатори_{opt} **event** тип делегата ідент. події;

event & **delegate** – елементи реалізації патерну **Observer**

Основа – підтримка делегатом списку зареєстрованих методів



EventHandler :

- стандартний системний делегат
- узагальнений системний делегат, параметр – тип даних події
- суфікс назви делегата, визначеного користувачем



EventArgs :

- стандартний системний **тип даних** події (без даних)
- суфікс назви типу даних події, похідного від **EventArgs**



Тип, який реагує на подію, надає метод-обробник події



Обробник події реєструється через делегат

Типи даних для подій EventArgs

```
[Serializable]
public class EventArgs
{
    public static readonly EventArgs Empty = new EventArgs();
    public EventArgs() { }
}
```

- ❑ .NET events are based on the **EventHandler** delegate and the **EventArgs** base class
- ❑ .NET provides many event data classes
- ❑ **EventArgs** – the base type for (not all) event data classes
- ❑ EventArgs is usually used when an event does not have any data associated with it and is only meant to notify other classes that something happened
- ❑ **EventArgs.Empty** value can be passed when no data is provided
- ❑ **EventHandler** delegate includes EventArgs class as a parameter
- ❑ Event receiver defines an event handler method to respond to an event

Системні делегати – типи обробітників подій

- Стандартний системний делегат

```
[SerializableAttribute]  
[ComVisibleAttribute(true)]  
public delegate void EventHandler (Object sender, EventArgs e )
```

- Узагальнений системний делегат, параметр – тип даних події

```
[SerializableAttribute]  
public delegate void EventHandler<TEventArgs> ( Object sender, TEventArgs e)  
    where TEventArgs : EventArgs
```

Events overview

- ❑ The **publisher** determines when an event is raised; the **subscribers** determine what action is taken in response to the event
- ❑ An event can have **multiple** subscribers. A subscriber can handle **multiple events** from multiple publishers
- ❑ Events that have no subscribers are never raised (? . using)
- ❑ Events are typically used to signal user actions such as button clicks or menu selections in **graphical user interfaces**
- ❑ When an event has multiple subscribers, the event handlers are invoked synchronously when an event is raised (using special approach to invoke events asynchronously)

Патерн Observer

❑ Назва та класифікація

Observer – патерн поведінки об'єктів

❑ Призначення

Визначає залежність “**один-до-багатьох**” між об'єктами, при якій у випадку зміни стану одного об'єкта залежні від нього об'єкти автоматично повідомляються про це і можуть оновлюватися

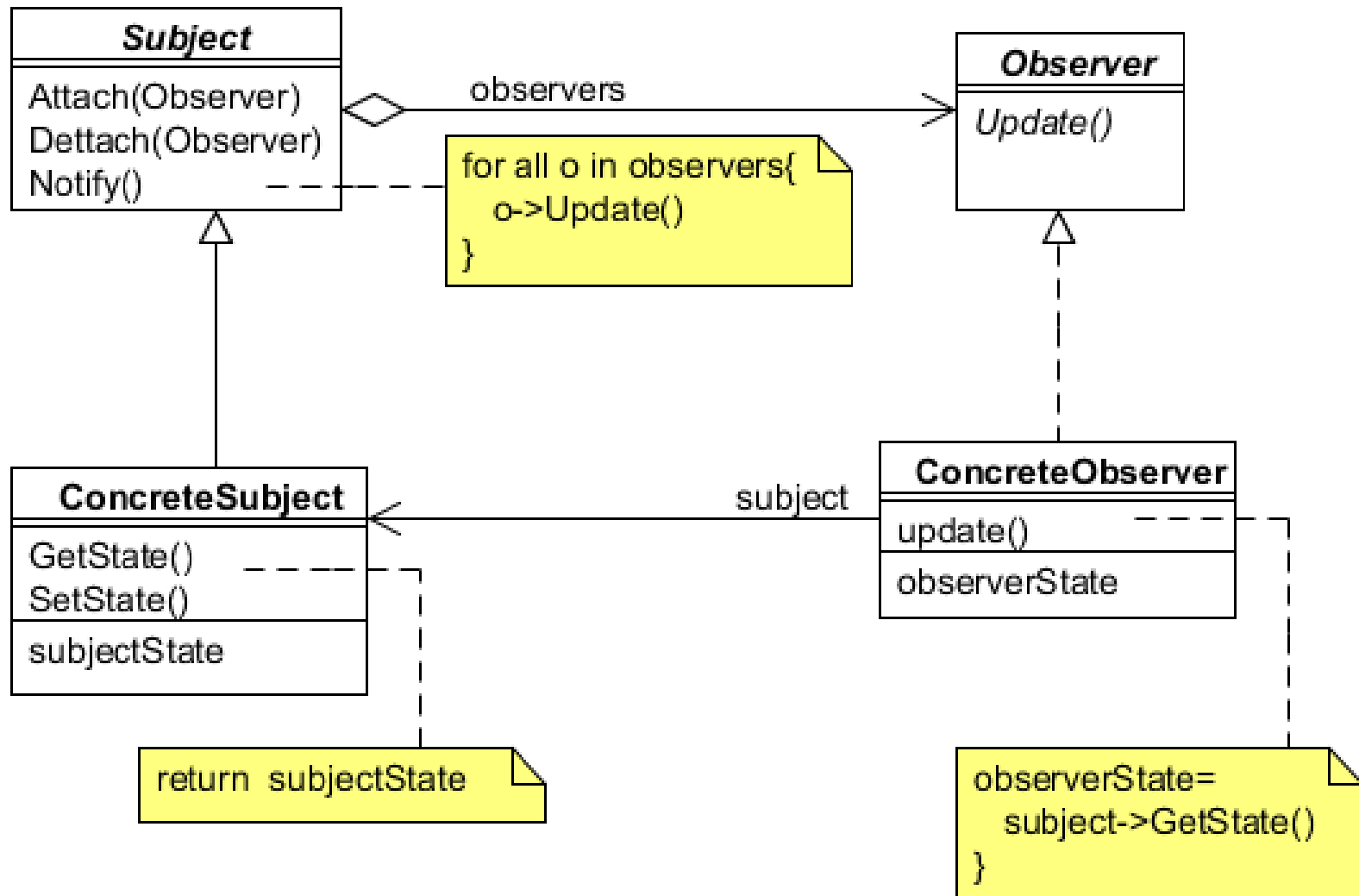
❑ Мотивація

- Патерн вказує спосіб узгодження колективної поведінки
- Використовують, щоб уникнути перетворення системи в **моноліт** (як побічний ефект розподілу системи на **взаємодіючі** класи): окремі класи важко повторно використовувати і складно змінювати алгоритм їхньої взаємодії

Observer Застосування

- Абстракція стосується двох окремих типів, один з яких залежить від іншого
- Модифікація стану одного об'єкта зумовлює зміну інших об'єктів
- Один об'єкт має сповістити про зміну свого стану інші об'єкти без жодних припущень щодо них
- Кількість залежних об'єктів може динамічно змінюватися
- Інкапсуляція у два різні об'єкти дає змогу незалежно їх змінювати і повторно використовувати

Observer



Observer Учасники

❑ Subject (Provider, Publisher)

- має в розпорядженні мінімальну інформацію (інтерфейс **Observer**) про своїх спостерігачів, кількість яких може динамічно змінюватися
- надає інтерфейс для приєднання і від'єднання спостерігачів

❑ Observer (Subscriber)

- визначає інтерфейс оновлення для об'єктів, які отримують повідомлення від суб'єкта про зміни його стану

❑ ConcreteSubject

- зберігає стан, параметрами якого цікавляться спостерігачі
- надає інформацію про параметри стану спостерігачам, коли відбувається зміна стану

❑ ConcreteObserver

- зберігає посилання на конкретного суб'єкта
- зберігає дані, які повинні узгоджуватися з суб'єктом
- реалізує інтерфейс **Observer** для оновлення узгоджених із суб'єктом даних

Observer C# реалізація

- ❑ **publisher**, який генерує подію – об'єкт **event** – повідомляє об'єкти-**subscribers** (які підписалися на отримання події) про те, що щось відбулося під час виконання програми
- ❑ **publisher** не знає про конкретні об'єкти, які підписалися на отримання події
- ❑ **subscriber** сам вирішує, як реагувати (обробляти) на подію, надаючи / вилучаючи операторами **+=** і **-=** для цього відповідний метод-обробітник

Observer Результати

❑ Абстрактна і слабка зв'язність суб'єкта і спостерігача:

- суб'єктові відомо, що він має спостерігачів з фіксованим інтерфейсом
- суб'єктові невідомо конкретний тип спостерігачів
- спостерігачі нових типів додаються без жодних змін коду суб'єкта; від нових типів спостерігачів вимагається лише реалізація інтерфейсу **Observer**
- суб'єкт і спостерігач можуть бути з різних рівнів своїх ієрархій

❑ Підтримка широкої смуги комунікації:

- повідомлення автоматично надсилається усім підписаним на нього спостерігачам, порядок їх до уваги братися не може
- кількість спостерігачів у довільний момент може незалежно змінюватися
- спостерігач сам вирішує, чи повинен він обробляти отримане повідомлення, чи може його ігнорувати

❑ Неочікувані оновлення:

- оскільки спостерігачі не мають інформації один про одного, тому не відомо, в що сумарно обходиться оновлення суб'єкта – операція над ним може спричинити цілий ряд оновлень спостерігачів та залежних від них об'єктів
- нечітко визначені критерії залежності можуть спричинити неочікувані оновлення, які складно відслідкувати

Observer Механізм залежностей



Архітектура відображення суб'єктів на спостерігачів

- суб'єкт володіє своїм станом і керує ним **один**; **кілька** спостерігачів використовують стан суб'єкта, але не володіють ним :

=> відображення **ОДИН-до-БАГАТЬОХ** між об'єктами

- суб'єкт зберігає явні посилання на спостерігачів, яким він надсилає повідомлення
- зростання накладних затрат при великій кількості суб'єктів і кількох спостерігачах



Спостереження за кількома суб'єктами інтерфейс `Update()` розширюють, напр., додаючи як **аргумент** посилання на суб'єкт



Висячі посилання на знищені суб'єкти перед знищенням суб'єкт має надіслати повідомлення про це спостерігачам (повідомити спостерігачів, що припиняє надсилати їм повідомлення)

Observer Механізм залежностей

❑ Хто ініціює оновлення: (2 варіанти)

- 1) операції об'єкта **Subject**, які змінили стан, викликають **Notify()**
 - перевага – спостерігачам не треба “пам'ятати” про те, щоб передбачити момент взаємодії з суб'єктом
 - недолік – повідомлення при виконанні кожної з послідовних операцій може привести до неефективного виконання програми
- 2) на клієнті (це може бути також додатковий об'єкт **Mediator**) відповідальність за своєчасний виклик **Notify()**
 - перевага – клієнт може відкласти ініціалізацію оновлення до завершення певної серії змін суб'єкта
 - недолік – додаткове навантаження на клієнта

Observer Механізм залежностей

❑ Гарантія несуперечливості стану суб'єкта перед надсиланням повідомлення:

- може порушуватися за рахунок виклику операцій базових класів
- вирішення – патерн **Template method** у абстрактному класі **Subject**, коли примітивна операція перевизначається у похідному класі

❑ Залежність протоколу оновлення:

- **push model:** суб'єкт надсилає спостерігачам незалежно від їхніх потреб детальну інформацію в повному обсязі про зміни свого стану :
 - акцент на інформованості типу **Subject** про тип спостерігачів **Observer**
 - імовірність повторного використання типу **Subject** зменшується, оскільки не можна передбачити усіх можливих потреб спостерігачів
- **pull model:** суб'єкт надсилає спостерігачам мінімальну інформацію лише про факт зміни свого стану
 - акцент на витягуванні спостерігачами додаткової інформації від суб'єкта про його новий стан
 - об'єкти типу **Observer** звертаються за потрібною інформацією для них до об'єкта **Subject**

Observer Механізм залежностей

- ❑ Явна специфікація модифікацій, які цікавлять спостерігача

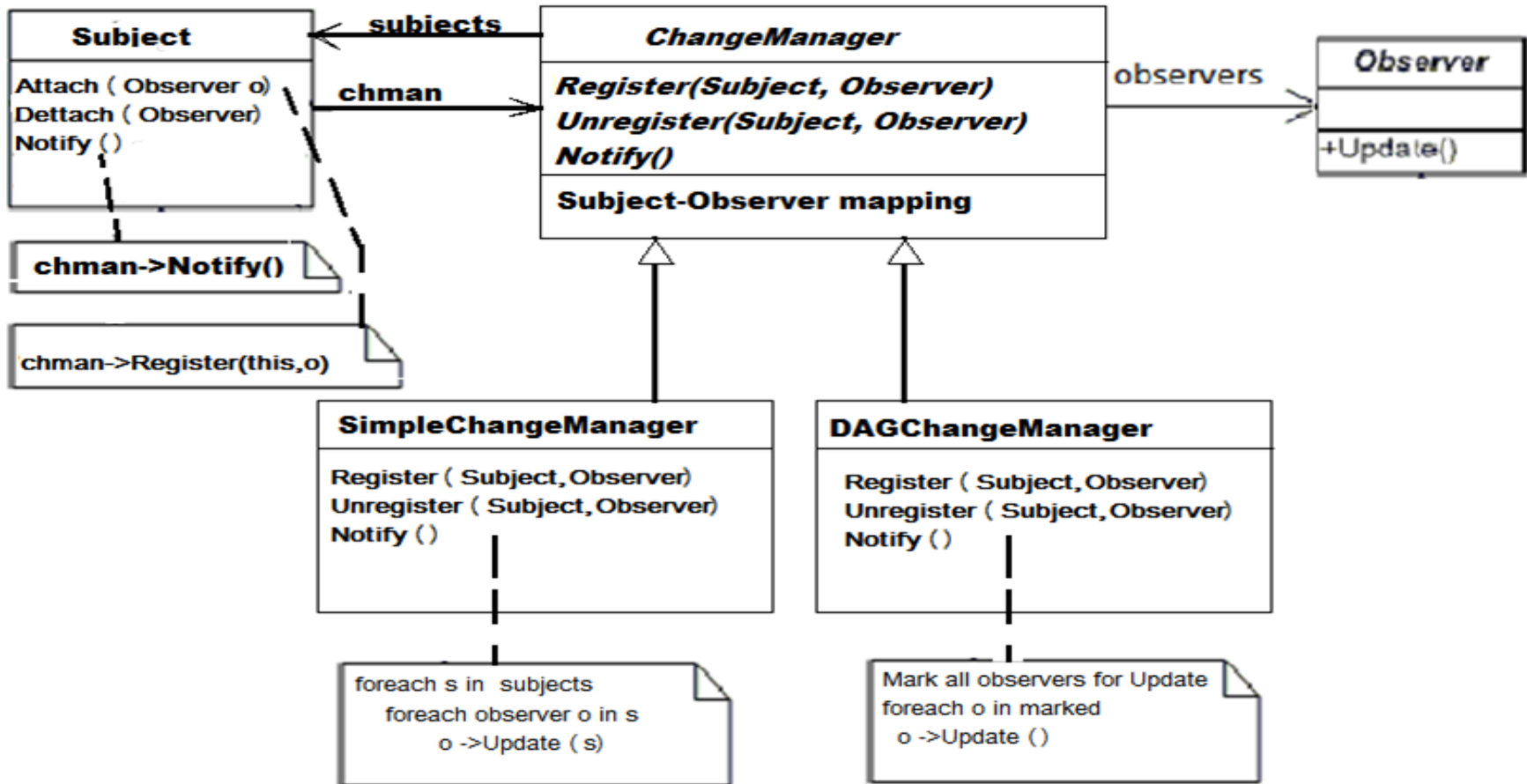
```
void Subject::Attach(Observer*, Aspect& interest);
```

```
void Observer::Update(Subject*, Aspect& interest);
```

Якщо зазначені модифікації відбудуться із суб'єктом, він посилатиме повідомлення лише тим спостерігачам, які проявили при реєстрації до них інтерес

- ❑ **Інкапсуляція складної семантики оновлення:** складне відношення між суб'єктом і спостерігачами організовуютьоруде
- ❑ згідно комбінації патернів **Mediator-Singleton**
 - будує відображення між суб'єктом і спостерігачами
 - додатковий об'єкт мінімізує кількість дій, необхідних для відображення спостерігачами змін у суб'єкті
 - надає інтерфейс для підтримки цього відображення в актуальному стані; це дає змогу суб'єктам позбутися посилань на своїх спостерігачів і навпаки
 - визначає конкретну стратегію оновлення
 - оновлює усіх залежних спостерігачів за запитом суб'єкта

Observer & Mediator



SimpleChangeManager оновлює залежних спостерігачів кожного суб'єкта

DAGChangeManager опрацьовує направлені ациклічні графи залежностей між суб'єктами та їхніми спостерігачами і гарантує, що при зміні кількох суб'єктів відповідний спостерігач отримає лише одне сповіщення

Event properties

- ❑ If class raises multiple events, the compiler generates one **field** per event delegate instance
- ❑ Event **property** can be used if :
 - the number of events is large – using a delegate collection that is indexed by an event key:
`public sealed class EventHandlerList : IDisposable`
 - class implements interfaces with the same event
- ❑ Event properties consist of event declarations accompanied by **event accessors** – methods, that **add** or **remove** event delegate instances from the storage data structure
- ❑ Event properties are slower than event fields, because each event delegate must be retrieved before it can be invoked

ImplementInterfaceEvents

WrapTwoInterfaceEvents

IObserver & IObservable

.NET Framework 4

```
public interface IObservable<in T>
{
    void OnNext( T value );
    void OnCompleted();
    void OnError( Exception error );
}

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

T represents the class that provides the notification information