

Делегати. Події

Клакович Л.М.

Делегати - Delegates

модифікатори *opt* **delegate** тип результату **Назва** (список формальних параметрів)

- **Делегат** використовується для створення об'єктів методів
- **Делегат** це спеціальний вид класу, який зберігає посилання на метод з наперед визначеною сигнатурою
- Метод Invoke викликає **делегат** як метод
- Ми можемо передавати **делегат** як аргумент в інший метод
- Використовується з подіями

```
delegate void PrintStringDelegate(string s);
```

```
delegate double FunctionDel(double x);
```

```
// оголошення делегату  
public delegate void PrintStringDelegate(string s);
```

```
class Example  
{  
    // метод виводить на консоль  
    public static void WriteToScreen(string str)  
    {  
        Console.WriteLine("The String is: {0}", str);  
    }  
  
    // метод виводить у файл  
    public static void WriteToFile(string s)  
    {  
        StreamWriter sw=new StreamWriter("message.txt",true);  
        sw.WriteLine(s);  
        sw.Close();  
    }  
}
```

- При створенні об'єкту типу делегат, ініціалізація можлива лише методом з вказаною делегатом сигнатурою (тип, що повертає і списком параметрів)

```
static void Main(string[] args)
{
    PrintStringDelegate ps1 = new PrintStringDelegate(Example.WriteToScreen);
    PrintStringDelegate ps2 = Example.WriteToFile;
    PrintStringDelegate ps3 = new PrintStringDelegate(c =>
        Console.WriteLine("Lambda is used! {0}", c));
    PrintStringDelegate ps4 = c=>
        Console.WriteLine("Simple lambda with delegate {0}", c);

    ps1("Hello!");
    ps2.Invoke("Hi!");
    ps3("Pryvit");
    ps4("Ogo");
}
```

- **Lambda вирази** надають простий спосіб визначення функцій

Передача делегатів, як аргументів в методи

```
delegate double FunctionDel(double x);
```

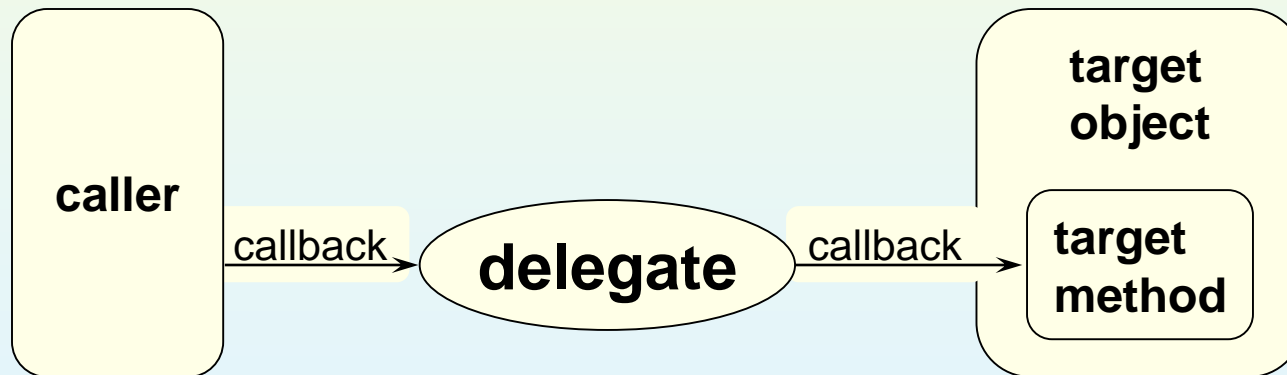
```
static void Tabulation(FunctionDel f, double a, double b, int n)
{
    Console.WriteLine("x : f(x)");
    double x = a;
    double h = (b - a) / n;
    for (int i = 0; i < n; i++)
    {
        Console.WriteLine("{0} : {1}", x, f(x));
        x = x + h;
    }
}
```

```
static void Main(string[] args)
{
    Tabulation(Math.Sqrt, 0, 10, 10);
    Tabulation(Fun, -2, 4, 10);
    Tabulation(c => c * 5 + 2, 0, 20, 10);
}
```

```
private static double Fun(double x){return x*x+4*x-5;}
```

Події і делегати

- **Події** використовують такі дії як натискання клавіші, кліки мишкою, зміна розміру вікна, закривання вікна і інші системні повідомлення.
- Програма, або її частина підписується на ці події і реагує на них через виклики своїх методів.
- Цю взаємодію незалежних частин забезпечують делегати.



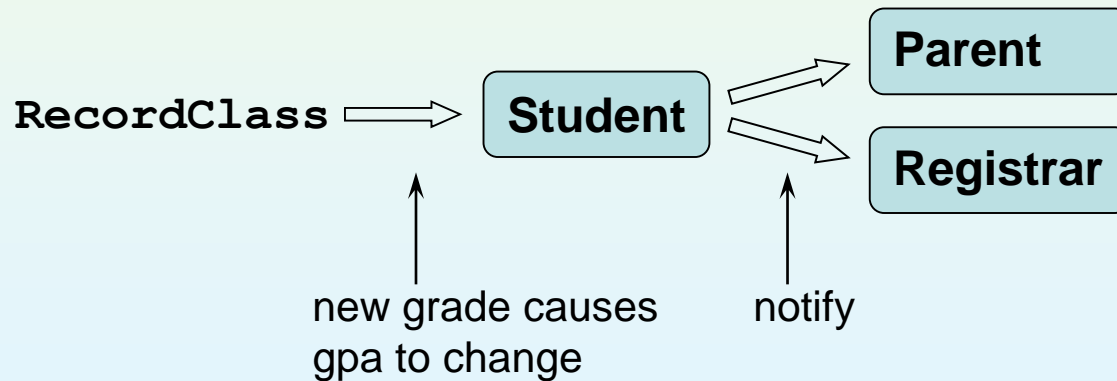
```
class Student
{
    string name;
    double gpa;
    int    units;

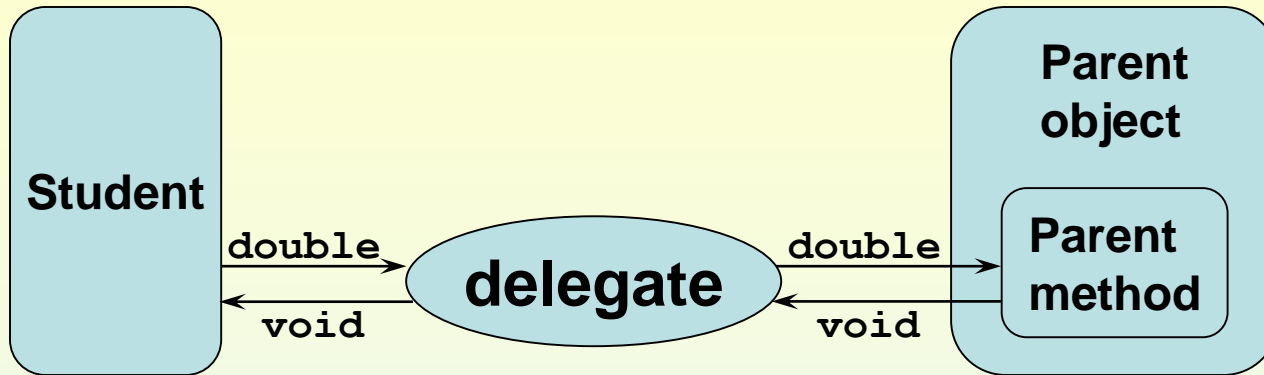
    public void Record(int grade)
    {
        gpa = (gpa * units + grade) / (units + 1);

        units++;
    }
    ...
}
```

store state →

change state →





delegate keyword

name of delegate

```
delegate void StudentCallback(double gpa);
```

target method
return type

target method
parameter

define delegate →

```
delegate void StudentCallback(double gpa);
```

target method →

```
class Parent  
{  
    public void Report(double gpa) { ... }  
}
```

caller stores delegate →

```
class Student  
{  
    public event StudentCallback GpaChanged;
```

caller invokes delegate →

```
    public void Record(int grade)  
    {  
        // update gpa  
        ...  
        GpaChanged(gpa);  
    }  
}
```

create and install delegate →

```
Student ann = new Student("Ann");  
Parent mom = new Parent();  
  
ann.GpaChanged = new StudentCallback(mom.Report);  
ann.Record(4);
```

Null reference

test before call

```
class Student
{
    public event StudentCallback GpaChanged;

    public void Record(int grade)
    {
        // update gpa
        ...
        if (GpaChanged != null)
            GpaChanged(gpa);
    }
}
```

Static methods

static method →

```
class Registrar
{
    public static void Log(double gpa)
    {
        ...
    }
}
```

register →

```
void Run()
{
    Student ann = new Student("Ann");

    ann.GpaChanged = new StudentCallback(Registrar.Log);
    ...
}
```

Multiple

- Можна комбінувати делегати використовуючи `operator+=` або `operator+`

targets →

```
Parent mom = new Parent();  
Parent dad = new Parent();
```

```
Student ann = new Student("Ann");
```

first →

```
ann.GpaChanged += new StudentCallback(mom.Report);
```

second →

```
ann.GpaChanged += new StudentCallback(dad.Report);
```

```
...
```

Remove delegate

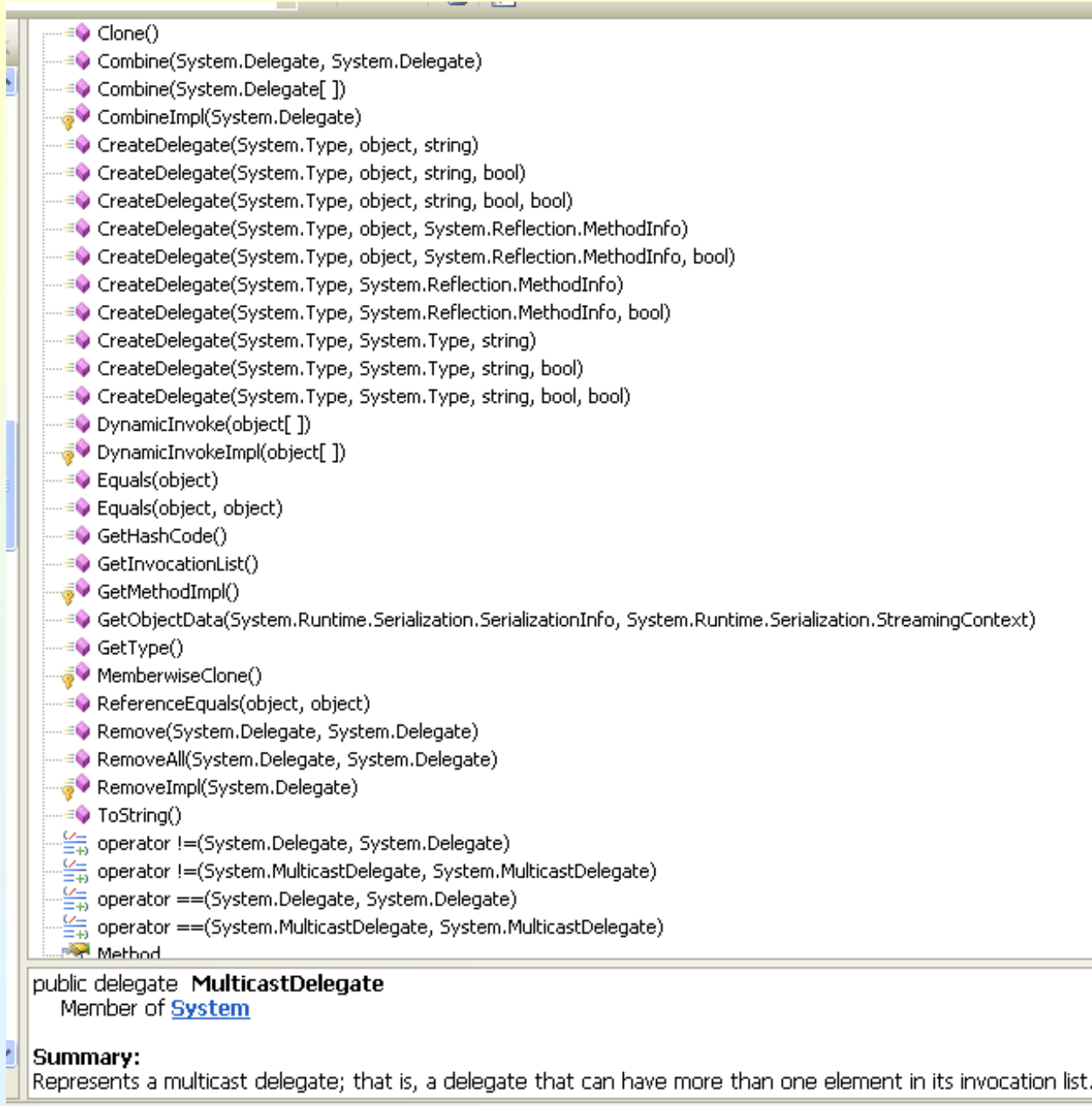
Можна видаляти делегати зі списку
використовуючи operator -= або operator-

```
Parent mom = new Parent();  
Parent dad = new Parent();  
  
Student ann = new Student("Ann");  
  
ann.GpaChanged += new StudentCallback(mom.Report);  
ann.GpaChanged += new StudentCallback(dad.Report);  
...  
ann.GpaChanged -= new StudentCallback(dad.Report);  
...
```

add →

remove →

public delegate MulticastDelegate



The screenshot displays the Visual Studio IDE with the `MulticastDelegate` class selected in the Solution Explorer. The class is a `public delegate` and is a member of the `System` namespace. The Methods window shows a list of methods, including `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()`, and operators `!=` and `==` for `System.Delegate` and `System.MulticastDelegate`. The Summary window at the bottom provides a brief description of the class.

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.

Події - Events

атрибути_{opt}

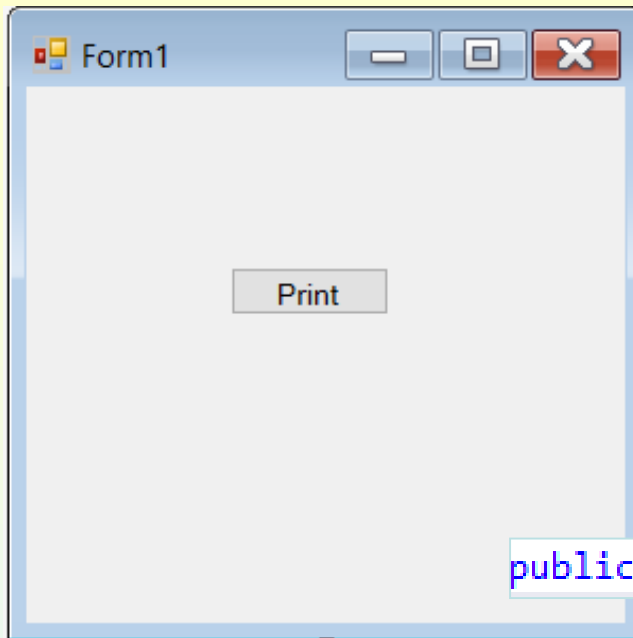
модифікатори_{opt} **event** тип делегата ідент. події ;

event & delegate – елементи реалізації патерну Observer

Основа – підтримка списку зареєстрованих делегатів

- **EventHandler** :
 - стандартний системний делегат
 - узагальнений системний делегат, аргумент -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 для обробки події
- Обробник події реєструється через делегат

Event Handler в WinForms



```
public class Control
```

```
{  
    public event EventHandler AutoSizeChanged;  
    public event EventHandler BackColorChanged;  
    public event EventHandler BackgroundImageChanged;  
    public event EventHandler BackgroundImageLayoutChanged;  
    public event EventHandler BindingContextChanged;  
    public event EventHandler CausesValidationChanged;  
    public event UIEventHandler ChangeUICues;  
    public event EventHandler Click;  
    public event EventHandler ClientSizeChanged;  
    public event EventHandler ContextMenuChanged;  
    public event EventHandler ContextMenuStripChanged;  
    public event ControlEventHandler ControlAdded;  
}
```

```
public delegate void EventHandler(object sender, EventArgs e);
```

```
public partial class Form1 : Form  
{  
    public Form1()  
    {  
        InitializeComponent();  
        this.button1.Click += new System.EventHandler(PrintManager.Print);  
    }  
}
```



```
static public class PrintManager
{
    static public void Print(object sender, EventArgs e)
    {
        using (StreamWriter file = new StreamWriter("result.txt"))
        {
            file.WriteLine("Document was printed");
        }
    }
}
```

Event Handler

Event handler – це делегат із специфічною сигнатурою

```
public delegate void MyEventHandler(object sender, MyEventArgs e);
```

Джерело події

Об'єкт, який містить деталі події

- Для реакції на подію потрібно визначити метод, який буде викликатися, в класі підписнику.
- Цей метод повинен мати сигнатуру, вказану делегатом цієї події.
- Для отримання повідомлення, коли настане подія, цей метод повинен бути підписаний на цю подію.

Підписка на подію Timer

```
static void Main(string[] args)
{
    Timer mytimer = new Timer(2000);
    mytimer.Elapsed += Print_Red;
    mytimer.Start();
    Console.ReadLine();
}

private static void Print_Red(object sender, ElapsedEventArgs e)
{
    Console.ForegroundColor = ConsoleColor.Red;
    Console.WriteLine("Elapsed: {0:HH:mm:ss}", e.SignalTime);
}
```

```
static void Main(string[] args)
{
    Timer mytimer = new Timer(2000);
    mytimer.Elapsed += Print_Red;
    mytimer.Elapsed += Print_White;
    mytimer.Start();
    Console.ReadLine();
}

private static void Print_White(object sender, ElapsedEventArgs e)
{
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine("Elapsed1: {0:HH:mm:ss}", e.SignalTime);
}

private static void Print_Red(object sender, ElapsedEventArgs e)
{
    Console.ForegroundColor = ConsoleColor.Red;
    Console.WriteLine("Elapsed: {0:HH:mm:ss}", e.SignalTime);
}
```

Відписка від події Timer

```
static void Main(string[] args)
{
    Timer mytimer = new Timer(2000);
    mytimer.Elapsed += Print_Red;
    mytimer.Elapsed += Print_White;
    mytimer.Start();
    Console.WriteLine("Press Enter to remove Red event");
    Console.ReadLine();
    mytimer.Elapsed -= Mytimer_Red;
    Console.ReadLine();
}
```