



DELEGATE & EVENTS





ĐẶT VẤN ĐỀ

- Sự kiện (event) trong lập trình: tương tác với chuột, bàn phím, ... → **sự kiện** phải gắn với **phương thức xử lý sự kiện**:
 - Phương thức tên gì?
 - 1 sự kiện có thể có nhiều phương thức xử lý được không?

→ DELEGATE

- Delegate là cầu nối trung gian giữa sự kiện và phương thức xử lý sự kiện (method handling);
- Chứa một danh sách các phương thức xử lý sự kiện và các phương thức sẽ lần lượt được gọi để thực thi khi sự kiện xảy ra.



KHÁI NIỆM DELEGATE

- Xem như là một kiểu mới – lớp đóng gói các phương thức (method signature):
- Đặc tính
 - An toàn kiểu;
 - Cơ chế hướng đối tượng.
- Delegate xem như lớp:
 - Có thể hiện;
 - Có thể chứa những tham chiếu đến 1 hay nhiều phương thức.



KHÁI NIỆM DELEGATE

- Có 2 khái niệm:
 - Kiểu delegate (delegate type);
 - Thực thể (Đối tượng) delegate (delegate instance).
- Một delegate định nghĩa một nguyên mẫu (signature):
 - Kiểu trả về;
 - Danh sách kiểu tham số.
- Tất cả các phương thức có cùng nguyên mẫu có thể được bổ sung vào thể hiện của delegate
- Thể hiện của Delegate chứa một danh sách các tham chiếu phương thức:
 - Có thể add (+) các method;
 - Có thể remove (-) các method.



KHÁI NIỆM DELEGATE

- Delegate là một kiểu dữ liệu tham chiếu (reference type) dùng để đóng gói một hay nhiều phương thức với tham số và kiểu trả về xác định.
- Cú pháp khai báo Delegate:
`modifer_access delegate kiểu_trả_về tên_delegate(danh_sách_tham_số);`
 - **modifer access**: private, public, protected, internal;
 - **kiểu_trả_về**: kiểu trả về của phương thức;
 - **tên_delegate**: tên của delegate;
 - **danh_sách_tham_số**: các tham số của phương thức.



KHỞI TẠO VÀ THỰC THI DELEGATE

- Khởi tạo Delegate:

Tên_delegate **Biến_delegate** = new **Tên_delegate**(**danh sách tham số**)

- Thực thi Delegate:

- Xem Delegate như một phương thức;
- Xem Delegate như là một tham số;
- Xem Delegate như một đối tượng (Sử dụng hàm Invoke).



KHỞI TẠO VÀ THỰC THI DELEGATE

- Define delegate:

```
public delegate void MyDelegate1(int x, int y)
```

Delegate cho dạng hàm:
void Method(int, int)

```
public delegate string MyDelegate2(float f)
```

Delegate cho dạng hàm:
string Method(float)



KHỞI TẠO VÀ THỰC THI DELEGATE

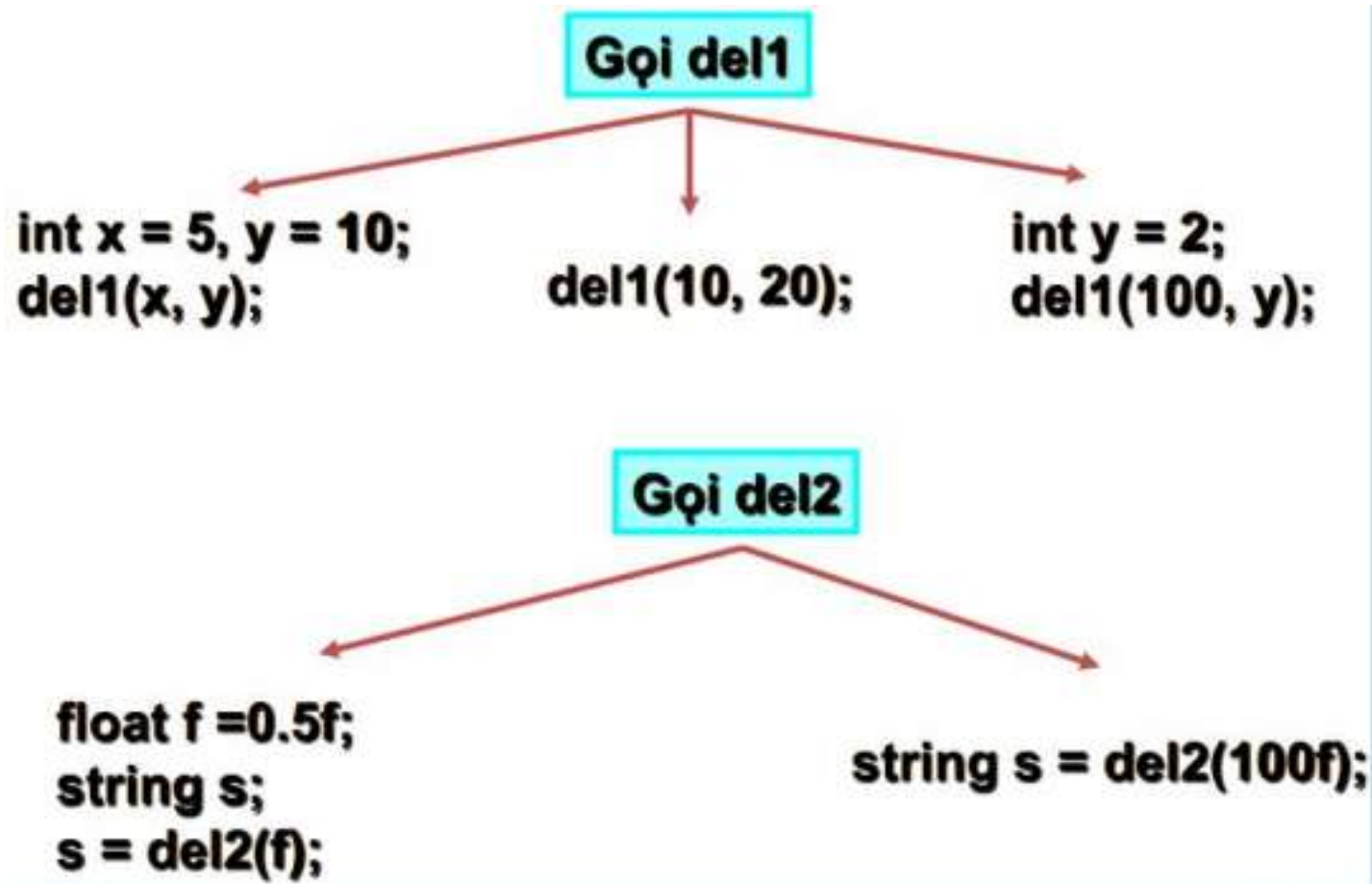
- Instance delegate:

```
public void Method1(int x, int y)
{
    ...
}
...
MyDelegate1 del1 = new MyDelegate1(Method1);

public string Method2(float f)
{
    ...
}
...
MyDelegate2 del2 = new MyDelegate2(Method2);
```


KHỞI TẠO VÀ THỰC THI DELEGATE

- Call delegate:





KHỞI TẠO VÀ THỰC THI DELEGATE

- Xem delegate như là một phương thức:

```
public delegate int Calculation(int a, int b);  
class Program  
{  
    static int Add(int a, int b){  
        return a + b;  
    }  
    static void Main(string[] args)  
    {  
        Calculation ca = new Calculation(Add);  
        int c = ca(5, 4);  
        Console.WriteLine("c = {0}", c);  
        Console.ReadLine();  
    }  
}
```



KHỞI TẠO VÀ THỰC THI DELEGATE

- Xem delegate như là một đối tượng:

```
public delegate int Calculation(int a, int b);  
class Program  
{  
    static int Add(int a, int b)  
    {  
        return a + b;  
    }  
    static void Main(string[] args)  
    {  
        Calculation ca = new Calculation(Add);  
        int kq = ca.Invoke(5, 4);  
        Console.WriteLine("Ket qua = {0}", kq);  
        Console.ReadLine();  
    }  
}
```



KHỞI TẠO VÀ THỰC THI DELEGATE

- Xem delegate như là một tham số:

```
public delegate int Calculation(int a, int b);
class Program
{
    static int Add(int a, int b)
    {
        return a + b;
    }
    static int Sub(int a, int b)
    {
        return (a - b);
    }
    static int Calculate(int a, int b, Calculation cal)
    {
        return cal(a, b);
    }
    static void Main(string[] args)
    {
        int c = Calculate(5, 4, Add);
        Console.WriteLine("c = {0}", c);
        int d = Calculate(5, 4, Sub);
        Console.WriteLine("d = {0}", d);
        Console.ReadLine();
    }
}
```



MULTICASTING

- Multicasting – một delegate có thể tham chiếu đến (gọi) nhiều phương thức:
 - Delegate phải có kiểu trả về là **void**;
 - Các phương thức này có cùng kiểu trả về và tham số với delegate.
- Thêm phương thức sử dụng toán tử “+”;
- Loại bỏ phương thức ra khỏi delegate sử dụng toán tử “-”.



MULTICASTING

```
void Print(int x,int y)
{
    Console.WriteLine("x = {0}, y = {1}", x, y);
}
void Sum(int x, int y)
{
    Console.WriteLine("Tong = {0}", x+y);
}

MyDelegate1 mulDel = new MyDelegate1(Print);
mulDel += new MyDelegate1(Sum);
mulDel(5, 10);
mulDel -= new MyDelegate1(Print);
mulDel(5,10);
```




ANONYMOUS METHOD

- Anonymous method: gắn delegate với các phương thức được khai báo tại thời điểm khởi tạo delegate → sạch hơn và thuận tiện trong khi mã hóa;
- Cú pháp:
<tên biến delegate> = delegate([tên các tham số tương ứng])
{ <Các dòng lệnh trong hàm>; };



HÀM SORT TỔNG QUÁT

- Xây dựng hàm SORT tổng quát cho mảng đối tượng có kiểu bất kỳ ???
- Trường hợp 1: Nếu đối tượng là kiểu số như int, long, float?
- Trường hợp 2: Nếu đối tượng phức tạp khác thì sẽ **so sánh như thế nào, theo quy tắc nào?**



HÀM SORT TỔNG QUÁT

- Sắp xếp trong các ngôn ngữ lập trình đã học: sử dụng 2 vòng lặp **for**:

```
for (int i=0; i< ...; i++)  
    for (int j=0; j< ...; j++)  
    {  
        ...your Algorithm ...  
    }
```

- Với C#: sử dụng method **Sort**:
 - Mảng là kiểu số;
 - Mảng là đối tượng phức hợp ???
- Sử dụng abstract method **CompareTo()** trong giao diện **IComparable**:

```
public int CompareTo(object obj);
```



HÀM SORT TỔNG QUÁT

- Giải pháp:
 - Cho phép đối tượng tự quy định thứ tự của chúng;
 - Sử dụng delegate để truyền phương thức so sánh này vào hàm Sort.

```
void Sort(object[] list, CompareObj cmp)
```

Delegate này sẽ tham chiếu tới hàm Compare của lớp MyClass. Chính lớp MyClass sẽ quy định thứ tự của các đối tượng



HÀM SORT TỔNG QUÁT

- Mô tả delegate CompareObj cho hàm Sort:

Tên của delegate

```
public delegate bool CompareObj(object o1,object o2)
```

Trả về true: nếu o1 “trước” o2
false: ngược lại

2 đối tượng cần so sánh



HÀM SORT TỔNG QUÁT

Định nghĩa hàm Sort tổng quát cho các lớp

Delegate sẽ trỏ tới hàm Compare riêng của lớp tương ứng

```
public static void Sort(object[] objs, CompareObj cmp)  
{
```

```
    for(int i=0; i < objs.Length-1; i++)  
        for(int j=objs.Length-1; j>i; j--)  
            if ( cmp( objs[j], objs[j-1] ) )  
            {  
                Swap( objs[j], objs[j-1] );  
            }  
}
```


Yêu cầu lớp tự so sánh



HÀM SORT TỔNG QUÁT

- Các lớp hỗ trợ Sort thì phải:
 - Cung cấp hàm Compare riêng;
 - Signature phải thoả delegate **CompareObj**.

```
class Person {  
    private string name;  
    private int weight;  
    private int yearOfBirth;  
    public static bool CompareName(object p1, object p2) {  
        if (string.Compare(((Person)p1).name, ((Person)p2).name) < 0)  
            return true;  
        return false;  
    }  
}
```



Cùng signature



HÀM SORT TỔNG QUÁT

```
public delegate bool CompareObj(object o1,object o2);
```

```
...
```

```
Person[ ] persons = new Person[4];
```

```
persons[0] = new Person("Quy Mui", 2, 2004);
```

```
persons[1] = new Person("Ha Lam", 65, 1978);
```

```
persons[2] = new Person("Ngoc Thao", 47, 1979);
```

```
persons[3] = new Person("Quoc Trung", 65, 1932);
```

```
CompareObj cmp = new CompareObj(Person.CompareName);
```

```
Lib.Sort( persons, cmp );
```

Gọi hàm Sort

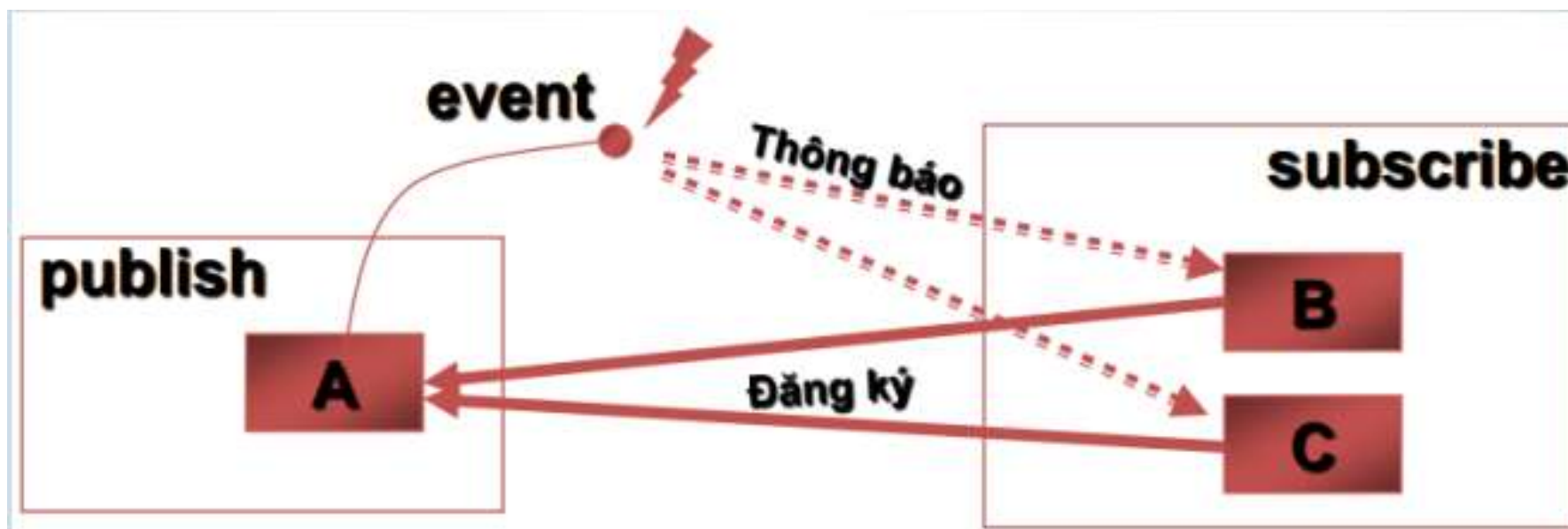
Lớp chứa hàm Sort

- Cơ chế thông điệp giữa các lớp hay các đối tượng;
- Có thể thông báo cho lớp khác biết được khi một lớp có phát sinh điều gì đó;
 - **Publisher**: lớp phát sinh sự kiện;
 - **Subscriber**: lớp nhận hay xử lý khi sự kiện xảy ra.
- Mỗi sự kiện thực chất là một delegate.



PUBLISHING & SUBSCRIBING

- Một lớp có publish một tập các event cho phép các lớp khác subscribe
 - Button là lớp publish đưa ra event: click;
 - Form là lớp subscribe có phần xử lý riêng khi “click” của Button kích hoạt.





EVENT & DELEGATE

- Sự kiện trong C# được thực thi nhờ uỷ quyền (delegate);
 - Lớp publishing định nghĩa sự uỷ quyền;
 - Những lớp subscribing phải thực thi;
 - Khi sự kiện xuất hiện thì phương thức của lớp subscribing được gọi thông qua delegate.
- Phương thức để xử lý sự kiện gọi là **trình xử lý sự kiện** (event handler).
- ❖ Lưu ý: phương thức xử lý sự kiện có thể không cần tuân theo nguyên mẫu của **trình xử lý sự kiện**.



EVENT & DELEGATE

- Trình xử lý sự kiện trong .NET được mô tả như sau:
 - Trả về giá trị **void**;
 - Tham số 1: nguồn phát sinh sự kiện, đây chính là đối tượng publisher (thường là kiểu liệu **object**);
 - Tham số 2: là đối tượng thuộc lớp **EventArgs** (hoặc lớp dẫn xuất từ EventArgs):
 - Lớp EventArgs là lớp cơ sở cho tất cả các dữ liệu về sự kiện: trừ hàm khởi tạo, lớp EventArgs kế thừa hầu hết các phương thức của lớp Object.



EVENT & DELEGATE

- Khai báo kiểu delegate xử lý sự kiện:

```
public delegate void HandlerName(object obj, EventArgs arg);
```

- Khai báo event gắn với kiểu delegate:

```
public event HandlerName OnEventName;
```

- Các lớp muốn xử lý khi sự kiện OnEventName phát sinh thì phải thực hiện quy trình xử lý sự kiện (event handler).



EVENT & DELEGATE

- Có 3 cách để đăng ký 1 event cho 1 đối tượng:
 - Tạo đối tượng delegate với mức truy cập là public;
 - Tạo đối tượng delegate với mức truy cập là private, và tạo property để truy xuất đến đối tượng delegate:
 - Có thể đăng ký nhiều phương thức cho sự kiện này được không?
 - Ví dụ: `someInstance.MyEvent += eventHandler; ???`
 - Tạo đối tượng delegate, sử dụng add và remove để thêm và xóa các đối tượng delegate.

❖ Lưu ý:

- Event có thể được khai báo trong interface, Delegate thì không;
- Event chỉ có thể được gọi (invoked) ở bên trong class chứa nó, Delegate có thể được gọi ở bất cứ đâu.



EVENT & DELEGATE

- Bài tập minh họa: Xây dựng 1 lớp thực hiện yêu cầu: “cứ mỗi giây sẽ phát sinh 1 sự kiện”:
- Cho phép 2 lớp khác đăng ký xử lý sự kiện này, mỗi lớp có cách xử lý riêng:
 - Lớp A: hiển thị thời gian theo **mô phỏng đồng hồ analog**;
 - Lớp B: hiển thị thời gian theo **mô phỏng đồng hồ digital**.

Thank You !

