

---

Bài 7: MỘT SỐ TÍNH NĂNG NÂNG CAO TRONG C# .....	2
1. Delegate (Ủy quyền) .....	2
1.1. Ủy quyền là gì? Khai báo, khởi tạo và gọi ủy quyền. ....	2
1.2. Sử dụng ủy quyền để xác nhận phương thức lúc thực thi.....	4
1.3. Multicast.....	11
1.4. Kiểu ủy quyền Func và Action .....	14
2. Anonymous method (phương thức vô danh) .....	16
3. Lambda expression.....	17
4. Anonymous type .....	19

## Bài 7: MỘT SỐ TÍNH NĂNG NÂNG CAO TRONG C#

### 1. Delegate (Ủy quyền)

#### 1.1. Ủy quyền là gì? Khai báo, khởi tạo và gọi ủy quyền.

**Ủy quyền** là 1 kiểu (type), đại diện cho các tham chiếu đến các phương thức có danh sách tham số và kiểu trả về cụ thể, tương tự như một con trỏ hàm trong C và C++.

- + Kiểu (type) của delegate được định nghĩa bởi tên của delegate.

#### Cú pháp khai báo ủy quyền

```
mức_truy_cập delegate kiểu_dữ_liệu_trả_về tên_ủy_quyền ([danh_sách_tham_số]);
```

- + Khai báo ủy quyền giống như khai báo phương thức, nhưng có thêm từ khóa delegate và không có thân phương thức
- + Câu lệnh khai báo ủy quyền phải đặt mức class hoặc name space, không đặt trong phương thức được.

Ví dụ sau khai báo một ủy quyền có tên là Del, có thể đóng gói một phương thức nhận một string làm đối số và trả về void:

```
public delegate void Del(string s);
```

#### Khởi tạo ủy quyền

Một đối tượng ủy quyền thường được khởi tạo bằng cách cung cấp tên của phương thức mà ủy quyền sẽ đóng gói, hoặc với một phương thức vô danh.

Giả sử ta có một phương thức tên là DelegateMethod có kiểu trả về là void và nhận 1 tham số kiểu string, tương đồng với kiểu ủy quyền Del

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

Ta khai báo ủy quyền del1

```
Del del1
```

Và khởi tạo ủy quyền bằng cách gán bằng phương thức DelegateMethod

```
del1 = DelegateMethod;
```

#### Gọi (call | invoke) ủy quyền

Ngay sau khi một ủy quyền được khởi tạo, một lời gọi phương thức được thực hiện tới ủy quyền sẽ được ủy quyền chuyển cho phương thức đó. Các tham số được lời gọi truyền cho ủy quyền sẽ được truyền cho phương thức và giá trị trả về, nếu có, từ phương thức được trả lại cho lời gọi bởi ủy quyền. Điều này được gọi là gọi ủy quyền (invoking the delegate).

Một ủy quyền được khởi tạo có thể được gọi như thể chính nó là phương thức được đóng gói.

Cách 1: thay vì gọi trực tiếp phương thức, ta gọi ủy quyền. Các tham số lời gọi truyền cho ủy quyền sẽ được truyền cho phương thức.

tên\_biến\_ủy\_quyền (danh\_sách\_tham\_số)

Ví dụ:

```
del1("; expected");
```

Cách 2: sử dụng phương thức Invoke của biến ủy quyền để gọi các phương thức lưu trong biến.

Ví dụ:

```
del1.Invoke("; expected");
```

*Chú ý*: khi biến ủy quyền nhận giá trị null, nếu ta thực thi ủy quyền sẽ phát sinh ngoại lệ. Vì vậy để an toàn trước khi thi hành ủy quyền ta nên kiểm tra đảm bảo biến khác null.

```
if (del1 != null)
    del1.Invoke("; expected");
```

hoặc

```
del1?.Invoke("; expected");
```

**Ví dụ 7.1:** Làm việc với ủy quyền

```
using System;

namespace DemoUyQuyen
{
    class Program
    {
        //1. Khai báo kiểu ủy quyền HienThiThongBao có thể tham chiếu đến
        //các phương thức không trả về dữ liệu và có 1 tham số kiểu string
        private delegate void HienThiThongBao(string thôngBao);
        static void Main(string[] args)
        {
            //2. Khai báo và khởi tạo biến ủy quyền del1
            HienThiThongBao del1 = ThôngBaoLoi;
            //3. Thực thi ủy quyền del1 chính là thực thi phương thức ThôngBaoLoi
            del1(" thiếu ;");
            Console.ReadLine();
        }

        //Khai báo phương thức tương đồng với ủy quyền HienThiThongBao
        //(tham số, kiểu trả về)
        private static void ThôngBaoLoi(string loi)
        {
            Console.ForegroundColor = ConsoleColor.Red;
            Console.WriteLine("\nChương trình của bạn có lỗi biên dịch sau: {0}",
```

```

    Console.ResetColor();
  }
}

```

 Kết quả:

Chương trình của bạn có lỗi biên dịch sau: thiếu ;

## 1.2. Sử dụng ủy quyền để xác nhận phương thức lúc thực thi

Ủy quyền có thể được sử dụng để xác định các phương thức tĩnh và các phương thức của thể hiện mà chúng ta không biết trước cho đến khi chương trình thực hiện.

Giả sử minh họa như sau, chúng ta muốn tạo một lớp chứa đơn giản gọi là Pair lớp này lưu giữ và sắp xếp hai đối tượng được truyền vào cho chúng. Tạm thời lúc này chúng ta cũng không thể biết loại đối tượng mà một Pair lưu giữ. Nhưng bằng cách tạo ra các phương thức bên trong các đối tượng này thực hiện việc sắp xếp và được ủy quyền, chúng ta có thể ủy quyền thực hiện việc sắp thứ tự cho chính bản thân của đối tượng đó.

Những đối tượng khác nhau thì sẽ sắp xếp khác nhau. Ví dụ, một Pair chứa các đối tượng đếm có thể được sắp xếp theo thứ tự số, trong khi đó một Pair nút lệnh button có thể được sắp theo thứ tự alphabe tên của chúng. Mong muốn của người tạo ra lớp Pair là những đối tượng bên trong của Pair phải có trách nhiệm cho biết thứ tự của chúng cái nào là thứ tự đầu tiên và cái nào là thứ hai. Để làm được điều này, chúng ta phải đảm bảo rằng các đối tượng bên trong Pair phải cung cấp một phương thức chỉ ra cho chúng ta biết cách sắp xếp các đối tượng.

Chúng ta định nghĩa phương thức yêu cầu bằng việc tạo một ủy quyền, ủy quyền này định nghĩa chữ ký (signature) và kiểu trả về của phương thức đối tượng để cung cấp và cho phép Pair xác định đối tượng nào đến đầu tiên và đối tượng nào là thứ hai.

Lớp Pair định nghĩa một ủy quyền, WhichIsFirst. Phương thức Sort sẽ lấy một tham số là thể hiện của WhichIsFirst. Khi một đối tượng Pair cần biết thứ tự của những đối tượng bên trong của nó thì nó sẽ yêu cầu ủy quyền truyền vào hai đối tượng chứa trong nó như là tham số. Trách nhiệm của việc xác định thứ tự của hai đối tượng được trao cho phương thức đóng gói bởi ủy quyền.

Để kiểm tra thực hiện cơ chế ủy quyền, chúng ta sẽ tạo ra hai lớp, lớp Cat và lớpStudent. Hai lớp này có ít điểm chung với nhau, ngoại trừ cả hai thực thi những phương thứcđược đóng gói bởi WhichIsFirst. Do vậy cả hai đối tượng này có thể được lưu giữ bên trong của đối tượng Pair.

Trong chương trình thử nghiệm, chúng ta sẽ tạo ra hai đối tượng Student và hai đối tượng Cat và lưu chúng vào mỗi một đối tượng Pair. Sau đó chúng ta sẽ tạo những đối tượng ủy quyền để đóng gói những phương thức của chúng, những phương thức này

phải phù hợp với chữ ký và kiểu trả về của ủy quyền. Sau cùng chúng ta sẽ yêu cầu những đối tượng Pair này sắp xếp những đối tượng Student và Cat, ta làm từng bước như sau:

Bắt đầu bằng việc tạo phương thức khởi tạo của lớp Pair, lấy hai đối tượng và đưa chúng vào trong mảng:

```
public class Pair
{
    // đưa vào 2 đối tượng theo thứ tự
    public Pair(object firstObject, object secondObject)
    {
        thePair[0] = firstObject;
        thePair[1] = secondObject;
    }

    // mảng lưu 2 đối tượng
    private object[] thePair = new object[2];
}
```

Tiếp theo là chúng ta phủ quyết phương thức ToString() để chứa giá trị mới của hai đối tượng mà Pair nắm giữ:

```
public override string ToString()
{
    // xuất thứ tự đối tượng thứ nhất trước đối tượng thứ hai
    return thePair[0].ToString() + "," + thePair[1].ToString();
}
```

Bây giờ thì chúng ta đã có hai đối tượng bên trong của Pair và chúng ta có thể xuất giá trị của chúng ra màn hình. Tiếp tục là chúng ta sẽ thực hiện việc sắp xếp và in kết quả sắp xếp

Hiện tại thì không xác định được loại đối tượng mà chúng ta có, do đó chúng ta sẽ ủy quyền quyết định thứ tự sắp xếp cho chính bản thân các đối tượng mà Pair lưu giữ bên trong. Do vậy, chúng ta yêu cầu rằng mỗi đối tượng được lưu giữ bên trong Pair thực hiện việc kiểm tra xem đối tượng nào sắp trước. Phương thức này lấy hai tham số đối tượng và trả về giá trị kiểu liệt kê: theFirstComeFirst nếu đối tượng đầu tiên được đến trước và theSecondComeFirst nếu giá trị thứ hai đến trước.

Những phương thức yêu cầu sẽ được đóng gói bởi ủy quyền WhichIsFirst được định nghĩa bên trong lớp Pair:

```
public delegate comparison WhichIsFirst(object obj1, object obj2);
```

Giá trị trả về là kiểu comparison đây là kiểu liệt kê:

```
public enum comparison
{
    theFirstComesFirst = 1,
    theSecondComesFirst = 2
}
```

Bất cứ phương thức tĩnh nào lấy hai tham số kiểu object và trả về kiểu comparison có thể được đóng gói bởi ủy quyền vào lúc thực thi.

Lúc này chúng ta định nghĩa phương thức Sort cho lớp Pair:

```
public void Sort(WhichIsFirst theDelegateFunc)
{
    if (theDelegateFunc(thePair[0], thePair[1]) ==
        comparison.theSecondComesFirst)
    {
        object temp = thePair[0];
        thePair[0] = thePair[1];
        thePair[1] = temp;
    }
}
```

Phương thức này lấy một tham số: một ủy quyền có kiểu WhichIsFirst với tên là theDelegateFunc. Phương thức Sort giao phó trách nhiệm quyết định thứ tự đến trước sau của hai đối tượng bên trong Pair đến phương thức được đóng gói bởi ủy quyền. Bên trong thân của Sort, phương thức ủy quyền được gọi và trả về một giá trị, giá trị này là một trong hai giá trị liệt kê của comparison.

Nếu giá trị trả về là theSecondComesFirst, đối tượng bên trong của Pair sẽ được hoán đổi vị trí, trường hợp ngược lại thì không làm gì cả.

Hãy tưởng tượng chúng ta đang sắp xếp những Student theo tên. Chúng ta viết một phương thức trả về theFirstComesFirst nếu tên của sinh viên đầu tiên đến trước và theSecondComesFirst nếu tên của sinh viên thứ hai đến trước. Nếu chúng ta đưa vào là “Amy, Beth” thì phương thức trả về kết quả là theFirstComesFirst. Và ngược lại nếu chúng ta truyền “Beth, Amy” thì kết quả trả về là theSecondComesFirst. Khi chúng ta nhận được kết quả theSecondComesFirst, phương thức Sort sẽ đảo hai đối tượng này trong mảng, và thiết lập là Amy ở vị trí đầu còn Beth ở vị trí thứ hai.

Tiếp theo chúng ta sẽ thêm một phương thức ReverseSort, phương thức này đặt các mục trong mảng theo thứ tự đảo ngược lại:

```
public void ReverseSort(WhichIsFirst theDelegateFunc)
{
    if (theDelegateFunc(thePair[0], thePair[1]) ==
        comparison.theFirstComesFirst)
    {
        object temp = thePair[0];
        thePair[0] = thePair[1];
        thePair[1] = temp;    }    }
```

Việc thực hiện cũng tương tự như phương thức Sort. Tuy nhiên, phương thức thực hiện việc hoán đổi nếu phương thức ủy quyền xác định là đối tượng trước tới trước. Do vậy, kết quả thực hiện của phương thức là đối tượng thứ hai sẽ đến trước. Lúc này nếu chúng ta truyền vào là “Amy, Beth”, phương thức ủy quyền sẽ trả về theFirstComesFirst, và

phương thức ReverseSort sẽ hoán đổi vị trí của hai đối tượng này, thiết lập Beth đến trước. Điều này cho phép chúng ta sử dụng cùng phương thức ủy quyền tương tự như Sort, mà không cần yêu cầu đối tượng hỗ trợ phương thức trả về giá trị được sắp ngược. Lúc này điều cần thiết là chúng ta tạo ra vài đối tượng để sắp xếp. Ta tạo hai lớp đối tượng đơn giản như sau: lớp đối tượng Student và lớp đối tượng Cat. Gán cho đối tượng Student một tên vào lúc tạo:

```
public class Student
{
    // lớp đối tượng Student
    public Student(string name)
    {
        this.name = name;
    }
}
```

Lớp đối tượng Student này yêu cầu hai phương thức, một là phương thức phủ quyết ToString(), và một phương thức khác được đóng gói như là phương thức ủy quyền. Lớp Student phải phủ quyết phương thức ToString() để cho phương thức ToString() của lớp Pair sử dụng một cách chính xác. Việc thực thi này thì không có gì phức tạp mà chỉ đơn thuần là trả về tên của sinh viên:

```
public override string ToString()
{
    return name;
}
```

Student cũng phải thực thi một phương thức hỗ trợ cho Pair.Sort() có thể ủy quyền xác định thứ tự của hai đối tượng xem đối tượng nào đến trước:

```
public static comparison WhichStudentComesFirst(Object o1, Object o2)
{
    Student s1 = (Student)o1;
    Student s2 = (Student)o2;
    return (String.Compare(s1.name, s2.name) < 0 ?
        comparison.theFirstComesFirst : comparison.theSecondComesFirst);
}
```

String.Compare là phương thức của .NET trong lớp String, phương thức này so sánh hai chuỗi và trả về một giá trị nhỏ hơn 0 nếu chuỗi đầu tiên nhỏ hơn chuỗi thứ hai và lớn hơn 0 nếu chuỗi thứ hai nhỏ hơn, và giá trị là 0 nếu hai chuỗi bằng nhau. Theo lý luận trên thì giá trị trả về là theFirstComesFirst chỉ khi chuỗi thứ nhất nhỏ hơn, nếu hai chuỗi bằng nhau hay chuỗi thứ hai lớn hơn, thì phương thức này sẽ trả về cùng giá trị là theSecondComesFirst.

Ghi chú rằng phương thức WhichStudentComesFirst lấy hai tham số kiểu đối tượng và trả về giá trị kiểu liệt kê comparison. Điều này để làm tương ứng và phù hợp với phương thức được ủy quyền Pair.WhichIsFirst.

Lớp thứ hai là Cat, để phục vụ cho mục đích của chúng ta, thì Cat sẽ được sắp xếp theo trọng lượng, nhẹ đến trước nặng. Ta có khai báo lớp Cat như sau:

```
public class Cat
{
    //lớp đối tượng Cat
    public Cat(int weight)
    {
        this.weight = weight;
    }

    // sắp theo thứ tự trọng lượng
    public static comparison WhichCatComesFirst(Object o1, Object o2)
    {
        Cat c1 = (Cat)o1;
        Cat c2 = (Cat)o2;
        return c1.weight > c2.weight ? comparison.theSecondComesFirst
                                     : comparison.theFirstComesFirst;
    }

    public override string ToString()
    {
        return weight.ToString();
    }

    // biến lưu trọng lượng
    private int weight;
}
```

Cũng tương tự như lớp Student thì lớp Cat cũng phủ quyết phương thức ToString() và thực thi một phương thức tĩnh với cú pháp tương ứng với phương thức ủy quyền. Và chúng ta cũng lưu ý là phương thức ủy quyền của Student và Cat là không cùng tên với nhau. Chúng ta không cần thiết phải làm cùng tên vì chúng ta sẽ gán đến phương thức ủy quyền lúc thực thi.

Ví dụ minh họa sau trình bày cách một phương thức ủy quyền được gọi.

### Ví dụ 7.2:

```
using System;

namespace DemoUyQuyềnCallback
{
    // khai báo kiểu liệt kê
    public enum comparison
    {
        theFirstComesFirst = 1,
        theSecondComesFirst = 2
    }

    // lớp Pair đơn giản lưu giữ 2 đối tượng
    public class Pair
    {
```



```
// khai báo ủy quyền
public delegate comparison WhichIsFirst(object obj1, object obj2);
// truyền hai đối tượng vào constructor
public Pair(object firstObject, object secondObject)
{
    thePair[0] = firstObject;
    thePair[1] = secondObject;
}
// phương thức sắp xếp thứ tự của hai đối tượng
// theo bất cứ tiêu chuẩn nào của đối tượng
public void Sort(WhichIsFirst theDelegateFunc)
{
    if (theDelegateFunc(thePair[0], thePair[1]) ==
        comparison.theSecondComesFirst)
    {
        object temp = thePair[0];
        thePair[0] = thePair[1];
        thePair[1] = temp;
    }
}
// phương thức sắp xếp hai đối tượng theo
// thứ tự nghịch đảo lại tiêu chuẩn sắp xếp
public void ReverseSort(WhichIsFirst theDelegateFunc)
{
    if (theDelegateFunc(thePair[0], thePair[1]) ==
        comparison.theFirstComesFirst)
    {
        object temp = thePair[0];
        thePair[0] = thePair[1];
        thePair[1] = temp;
    }
}

// yêu cầu hai đối tượng đưa ra giá trị của nó
public override string ToString()
{
    return thePair[0].ToString() + "," + thePair[1].ToString();
}

// mảng lưu 2 đối tượng
private object[] thePair = new object[2];
}

public class Cat
{
    //lớp đối tượng Cat
    public Cat(int weight)
    {
        this.weight = weight;
    }
}
```

```
// sắp theo thứ tự trọng lượng
public static comparison WhichCatComesFirst(Object o1, Object o2)
{
    Cat c1 = (Cat)o1;
    Cat c2 = (Cat)o2;
    return c1.weight > c2.weight ? comparison.theSecondComesFirst
                                   : comparison.theFirstComesFirst;
}

public override string ToString()
{
    return weight.ToString();
}

// biến lưu trọng lượng
private int weight;
}

public class Student
{
    // lớp đối tượng Student
    public Student(string name)
    {
        this.name = name;
    }
    // sắp theo thứ tự chữ cái
    public static comparison WhichStudentComesFirst(Object o1, Object o2)
    {
        Student s1 = (Student)o1;
        Student s2 = (Student)o2;
        return (String.Compare(s1.name, s2.name) < 0 ?
            comparison.theFirstComesFirst : comparison.theSecondComesFirst);
    }
    public override string ToString()
    {
        return name;
    }
    // biến lưu tên
    private string name;
}

class Program
{
    public static void Main()
    {
        // tạo ra hai đối tượng Student và Cat
        // đưa chúng vào hai đối tượng Pair
        Student Thao = new Student("Thao");
        Student Ba = new Student("Ba");

        Cat Mun = new Cat(5);
        Cat Ngao = new Cat(2);
    }
}
```

```

Pair studentPair = new Pair(Thao, Ba);
Pair catPair = new Pair(Mun, Ngao);

Console.WriteLine("Sinh vien \t\t\t: {0}", studentPair.ToString());
Console.WriteLine("Meo \t\t\t: {0}", catPair.ToString());

// tạo ủy quyền
Pair.WhichIsFirst theStudentDelegate = new
    Pair.WhichIsFirst(Student.WhichStudentComesFirst);
Pair.WhichIsFirst theCatDelegate = new
    Pair.WhichIsFirst(Cat.WhichCatComesFirst);

// sắp xếp dùng ủy quyền
studentPair.Sort(theStudentDelegate);
Console.WriteLine("Sau khi sap xep studentPair\t\t:{0}",
    studentPair.ToString());

studentPair.ReverseSort(theStudentDelegate);
Console.WriteLine("Sau khi sap xep nguoc studentPair\t\t:{0}",
    studentPair.ToString());

catPair.Sort(theCatDelegate);
Console.WriteLine("Sau khi sap xep catPair\t\t:{0}",
    catPair.ToString());

catPair.ReverseSort(theCatDelegate);
Console.WriteLine("Sau khi sap xep nguoc catPair\t\t:{0}",
    catPair.ToString());

Console.ReadLine();
    }
}
}

```



Kết quả:

Sinh vien : Thao,Ba

Meo : 5,2

Sau khi sap xep studentPair :Ba,Thao

Sau khi sap xep nguoc studentPair :Thao,Ba

Sau khi sap xep catPair :2,5

Sau khi sap xep nguoc catPair :5,2

### 1.3. Multicast

Cơ chế multicast cho phép gọi nhiều phương thức thực thi thông qua một ủy quyền đơn. Điều này hoàn toàn khác với việc có một tập hợp các ủy quyền, vì mỗi trong số chúng chỉ gọi được duy nhất một phương thức.

Ủy quyền Multicast có thể được kết hợp từ các ủy quyền khác bằng phép toán cộng (+). Kết quả là một ủy quyền Multicast mới và gọi đến tất cả các phương thức thực thi nguyên thủy của cả hai bên.

Ví dụ, giả sử del1 và del2 là các thể hiện của 1 kiểu ủy quyền, dòng lệnh theo sau sẽ kết hợp chúng lại với nhau và tạo ra một ủy quyền Multicast mới:

```
multiDel = del1 + del2
```

Chúng ta cũng có thể thêm những ủy quyền vào trong ủy quyền Multicast bằng toán tử cộng bằng (+=).

Phép toán này sẽ thêm ủy quyền ở phía bên phải của toán tử vào ủy quyền Multicast ở bên trái.

Ví dụ, giả sử có del3 và multiDel là những ủy quyền, lệnh tiếp theo sau đây sẽ thực hiện việc thêm ủy quyền del3 vào trong multiDel:

```
multiDel += del3;
```

Để xóa 1 ủy quyền hoặc xóa 1 phương thức khỏi danh sách gọi của ủy quyền ta sử dụng toán tử - hoặc -=

Ví dụ, xóa ủy quyền del2 khỏi danh sách gọi của multiDel .

```
multiDel -= del2;
```

### Ví dụ 7.3: Kết hợp các ủy quyền

```
using System;

namespace DemoMulticast
{
    class Program
    {
        // Khai báo ủy quyền
        private delegate void HienThiThongBao(string thongBao);
        static void Main(string[] args)
        {
            // Khởi tạo ủy quyền del1, đóng gói phương thức ThongBaoLoi
            HienThiThongBao del1 = ThongBaoLoi;

            // Khởi tạo ủy quyền del2, đóng gói phương thức GuiThongDiep
            HienThiThongBao del2 = GuiThongDiep;

            //Khai báo ủy quyền multicast
            HienThiThongBao multiDel;

            //Khởi tạo ủy quyền multicast
            multiDel = del1 + del2;
            //Sau khi thực hiện lệnh multiDel = del1 + del2; danh sách
            //gọi của ủy quyền multiDel gồm có 2 phương thức là
            //ThongBaoLoi và GuiThongDiep
        }
    }
}
```

```

    HienThiThongBao del3 = CanhBao;
    multiDel += del3;

    //Sau khi thực hiện lệnh multiDel += del3; danh sách gọi của
    //ủy quyền multiDel gồm có 3 phương thức là ThongBaoLoi và
    //GuiThongDiep và CanhBao

    multiDel += CanhBao;
    multiDel += CanhBao;
    //Sau khi thực hiện lệnh multiDel += CanhBao; 2 lần ủy quyền
    //multiDel sẽ gọi phương thức ThongBaoLoi 1 lần, GuiThongDiep
    //1 lần, và CanhBao 3 lần

    multiDel -= del2;
    //Sau khi thực hiện lệnh multiDel -= del2; ủy quyền multiDel
    // sẽ gọi phương thức ThongBaoLoi 1 lần và CanhBao 3 lần,
    // phương thức GuiThongDiep đã bị xóa khỏi danh sách gọi
    multiDel("ABC");

    Console.ReadLine();
}

//Khai báo phương thức tương đồng với ủy quyền HienThiThongBao
//(tham số, kiểu trả về)
private static void ThongBaoLoi(string loi)
{
    Console.ForegroundColor = ConsoleColor.Red;
    Console.WriteLine("\nChương trình của bạn có lỗi biên dịch sau: {0}",
                                                                loi);
    Console.ResetColor();
}

//phương thức cũng có signature tương đồng với ủy quyền
static void GuiThongDiep(string ten)
{
    Console.WriteLine("\nThông điệp đã được gửi cho: {0}", ten);
}

//phương thức cũng có signature tương đồng với ủy quyền
static void CanhBao(string phienBan)
{
    Console.ForegroundColor = ConsoleColor.Yellow;
    Console.WriteLine("\nBạn nên dùng phiên bản {0}", phienBan);
    Console.ResetColor();
}
}
}

```



Kết quả:

Chương trình của bạn có lỗi biên dịch sau: ABC

Bạn nên dùng phiên bản ABC

Bạn nên dùng phiên bản ABC

Bạn nên dùng phiên bản ABC

## 1.4. Kiểu ủy quyền Func và Action

### 1.3.1. Func

Là các kiểu ủy quyền tương ứng với các phương thức có trả về dữ liệu.

#### Cú pháp khai báo đối tượng ủy quyền dùng Func

`Func<kiểu_tham_số_1, kiểu_tham_số_2, ..., kiểu_trả_về>` biến\_ủy\_quyền;

+ Kiểu trả về của phương thức trong khai báo Func bắt buộc phải có.

+ Số lượng tham số đầu vào từ 0 - 16

**Ví dụ:** Khai báo biến ủy quyền del1, tham chiếu đến phương thức có 2 tham số đầu vào có kiểu int và string, phương thức trả về kiểu bool.

```
Func<int, string, bool> del1;
```

- Nếu dùng cách thông thường, khai báo trên tương ứng với

```
//Khai báo ủy quyền ở lớp
delegate bool TenUyQuyền(int ts1, string ts2);
//Khai báo biến ủy quyền trong phương thức
TenUyQuyền del1;
```

### 1.3.2. Action

Là các kiểu ủy quyền tương ứng với các phương thức không trả về dữ liệu (kiểu trả về là void).

#### Cú pháp khai báo đối tượng ủy quyền dùng Action

`Action<kiểu_tham_số_1, kiểu_tham_số_2, ...>` biến\_ủy\_quyền;

Số lượng tham số đầu vào từ 1 - 16

**Ví dụ:** Khai báo biến ủy quyền del2, tham chiếu đến phương thức có 2 tham số đầu vào có kiểu double và string, phương thức không trả về dữ liệu.

```
Action<double, string> del2;
```

- Nếu dùng cách thông thường, khai báo trên tương ứng với

```
//Khai báo ủy quyền trong lớp
delegate void TenUyQuyền(double ts1, string ts2);
//Khai báo biến ủy quyền
TenUyQuyền del2;
```

#### Ví dụ 7.4: Sử dụng Func và Action

```
using System;

namespace Demo_Func_Action
```

```
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("\nSU DUNG FUNC");
            //Khai báo biến ủy quyền dùng Function
            Func<int, int, int> del1 = Cong2So;
            //In ra kết quả tính toán của 2 số, ví dụ 3 và 5
            Console.WriteLine("\n{0} + {1} = {2}", 3, 5, del1(3, 5));

            Console.WriteLine("\nSU DUNG ACTION");
            //Khai báo ủy quyền dùng Action
            Action<double> del2 = XepLoaiHocSinh;
            //In ra xếp loại của học sinh, ví dụ điểm trung bình 7.5
            del2(7.5);

            Console.ReadLine();
        }

        //phương thức có tham số và kiểu trả về tương đồng với ủy quyền
        //được khai báo bằng Func
        private static int Cong2So(int a, int b)
        {
            return a + b;
        }

        //phương thức có tham số và kiểu trả về tương đồng với ủy quyền
        //được khai báo bằng Action
        static void XepLoaiHocSinh(double diemTB)
        {
            if (diemTB >= 8)
                Console.WriteLine("\nHoc sinh dat loai: Gioi");
            else if (diemTB >= 6.5)
                Console.WriteLine("\nHoc sinh dat loai: Kha");
            else if (diemTB >= 5)
                Console.WriteLine("\nHoc sinh dat loai: Trung binh");
            else if (diemTB >= 3.5)
                Console.WriteLine("\nHoc sinh dat loai: Yeu");
            else
                Console.WriteLine("\nHoc sinh dat loai: Kem");
        }
    }
}
```



Kết quả:

SU DUNG FUNC

3 + 5 = 8

SU DUNG ACTION

---

Học sinh dat loại: Kha

---

## 2. Anonymous method (phương thức vô danh)


- Phương thức vô danh là các phương thức không có tên.
- Được khai báo bằng việc tạo thể hiện của ủy quyền cho phương thức đó, sử dụng toán tử delegate
- Kiểu dữ liệu trả về của 1 phương thức vô danh được suy ra từ lệnh return bên trong thân của phương thức.

Ví dụ:

```
Func<int, int, int> sum = delegate (int a, int b)
{
    return a + b;
};

Console.WriteLine(sum(3, 4));
```

---

 Kết quả:

7

---

Khi sử dụng toán tử delegate , ta có thể bỏ qua danh sách tham số. Nếu làm điều đó, phương thức vô danh đã tạo có thể được chuyển đổi thành kiểu ủy quyền với bất kỳ danh sách tham số nào, như trong ví dụ sau:

```
Action greet = delegate {
    Console.WriteLine("Hello!");
};

greet();
Action<int, double> introduce =
    delegate {
        Console.WriteLine("This is world!");
    };

introduce(42, 2.7);
```

---

 Kết quả:

Hello!  
This is world!

---



### 3. Lambda expression

Biểu thức lambda được sử dụng để tạo một phương thức vô danh. Sử dụng toán tử lambda => để tách danh sách tham số của lambda khỏi phần thân của nó. Biểu thức lambda có thể có hai dạng: Expression lambda và Statement lambda

#### 3.1. Expression lambda:

Một biểu thức lambda có một biểu thức bên phải toán tử => được gọi là expression lambda. Một expression lambda trả về kết quả của biểu thức và có cấu trúc cơ bản sau:

(**input-parameters**) => expression

Thân của biểu thức lambda có thể chứa lời gọi phương thức.

#### 3.2. Statement lambda

Một statement lambda giống một Expression lambda, nhưng các câu lệnh của nó được đặt trong cặp dấu { }

(**input-parameters**) => { <sequence-of-statements> }

Phần thân của một statement lambda có thể bao gồm bất kỳ số lượng câu lệnh nào.

Ví dụ:

```
Action<string> greet = name =>
{
    string greeting = $"Hello {name}!";
    Console.WriteLine(greeting);
};
greet("World");
```



Kết quả:

Hello World!

#### 3.3. Các tham số đầu vào (input parameters) của một biểu thức lambda

Các tham số đầu vào của một biểu thức lambda đặt trong cặp dấu ngoặc đơn.

Chỉ định không có tham số đầu vào bằng dấu ngoặc đơn ( ):

```
Action line = () => Console.WriteLine();
```

Nếu biểu thức lambda chỉ có một tham số, cặp dấu ( ) là tùy chọn:

```
Func<double, double> cube = x => x * x * x;
```

Nếu có hai hoặc nhiều tham số hơn, các tham số được phân cách bởi dấu ,

```
Func<int, string, bool> isTooLong = (int x, string s) => s.Length > x;
```

Có thể không cần khai báo kiểu của các tham số

```
Func<int, int, bool> testForEquality = (x, y) => x == y;
```

Tất cả các tham số đầu vào phải được khai báo kiểu tường minh hoặc ngầm định, nếu không sẽ có lỗi biên dịch

**Ví dụ 7.5:** phương thức vô danh và biểu thức lambda

```
using System;

namespace Demo_Lambda
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("\nSu dung FUNC va PHUONG THUC VO DANH");
            //Khai báo ủy quyền dùng Function
            Func<int, int, int> del1;
            //Khởi tạo ủy quyền dùng phương thức vô danh
            del1 = delegate (int x, int y)
            {
                return x + y;
            };
            //In ra kết quả tính toán của 2 số, ví dụ 2 và 4
            Console.WriteLine("\n{0} + {1} = {2}", 2, 4, del1(2, 4));

            Console.WriteLine("\nSu dung ACTION va BIEU THUC LAMBDA");
            //Khai báo ủy quyền dùng Action
            Action<double> del2;
            //Khởi tạo ủy quyền dùng biểu thức lambda
            del2 = (double diemTB) =>
            {
                if (diemTB >= 8)
                    Console.WriteLine("\nHoc sinh dat loai: Gioi");
                else if (diemTB >= 6.5)
                    Console.WriteLine("\nHoc sinh dat loai: Kha");
                else if (diemTB >= 5)
                    Console.WriteLine("\nHoc sinh dat loai: Trung binh");
                else if (diemTB >= 3.5)
                    Console.WriteLine("\nHoc sinh dat loai: Yeu");
                else
                    Console.WriteLine("\nHoc sinh dat loai: Kem");
            };

            //In ra xếp loại của học sinh, ví dụ điểm trung bình 9
            del2(9);

            Console.ReadLine();
        }
    }
}
```

 Kết quả:

Su dung FUNC va PHUONG THUC VO DANH

$2 + 4 = 6$

Su dung ACTION va BIEU THUC LAMBDA

Hoc sinh dat loai: Gioi

-----

## 4. Anonymous type

**Anonymous type** (kiểu vô danh) cho phép dễ dàng đóng gói một tập các thuộc tính chỉ đọc vào một đối tượng duy nhất mà trước đó không cần phải xác định tường minh kiểu (type) của nó.

- + Tên kiểu được tạo bởi trình biên dịch và không có sẵn ở cấp mã nguồn.
- + Kiểu của mỗi thuộc tính được suy ra bởi trình biên dịch
- + Tạo kiểu vô danh bằng cách sử dụng toán tử new cùng với việc khởi tạo một đối tượng.

Cú pháp:

```
var tên-đối-tượng = new { tên-thuộc-tính = giá-trị [, . . .] };
```

Ví dụ tạo đối tượng kiểu vô danh có ba thuộc tính là mã sách, tên sách và giá

```
var sach = new
{
    MaSach = "S01",
    TenSach = "Lập trình c#",
    Gia = 100
};
```

Kiểu vô danh được sử dụng nhiều trong các truy vấn LINQ

Chú ý:

- + Thuộc tính của đối tượng vô danh là chỉ đọc . Vì vậy nếu ta cố gắng gán 1 giá trị cho thuộc tính của đối tượng thì sẽ bị lỗi.
- + Để truy cập thuộc tính của đối tượng vô danh ta viết: tên-đối-tượng. tên-thuộc-tính.

Ví dụ: in ra tên của đối tượng sách

```
Console.WriteLine(sach.TenSach);
```