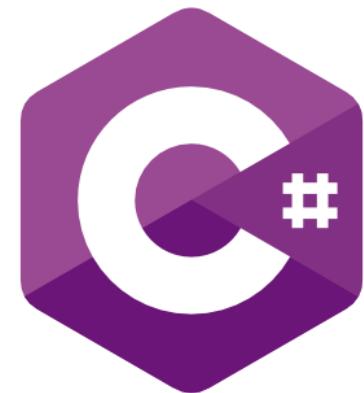


C# 12 – Event, Delegate và Collection

Giảng viên: **ThS. Lê Thiện Nhật Quang**
Email: quangltn.dotnet.vn@gmail.com
Website: <http://dotnet.edu.vn>
Điện thoại: **0868.917.786**

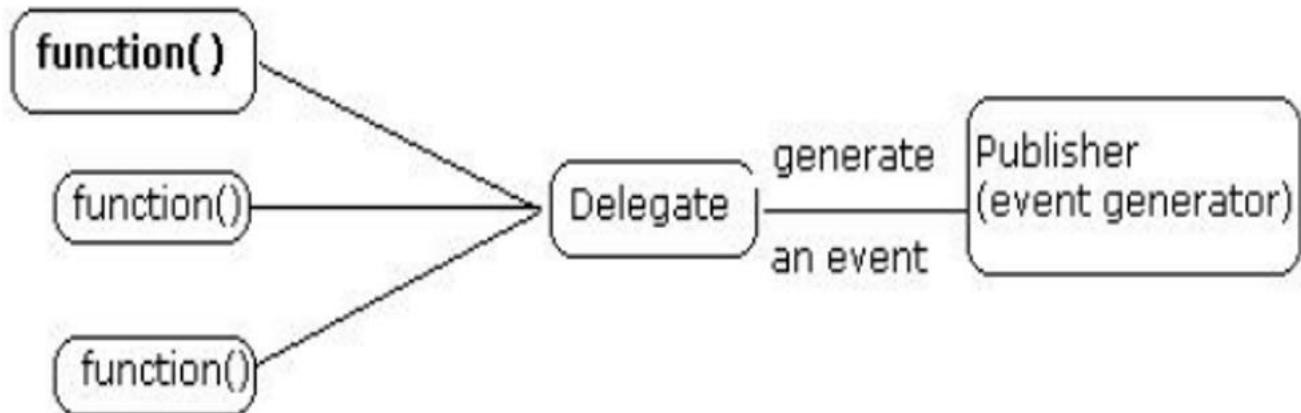


MỤC TIÊU

- Giải thích Delegate - ủy nhiệm
- Giải thích Event - sự kiện
- Tìm hiểu Collection – Bộ tập hợp

1.1. DELEGATE

- ❑ Delegate là một đối tượng chứa **tham chiếu** đến phương thức cần thực thi.
- ❑ Một delegate có thể trả tới một hoặc nhiều phương thức
- ❑ Delegate có thể gọi bất kỳ phương thức nào nó trả tới tại thời điểm thực thi.
- ❑ Tham chiếu của Delegate có thể thay đổi runtime (khi chương trình đang thực thi).
- ❑ Delegate thường được dùng để triển khai các phương thức hoặc sự kiện callback
- ❑ Delegate là một biến bình thường, biến này chứa hàm mà bạn cần gọi. Sau này lôi ra sài như hàm bình thường. Giá trị của biến Delegate lúc này là tham chiếu đến hàm. Có thể thay đổi runtime khi chương trình đang chạy.



1.2. KHAI BÁO DELEGATE

Khai báo Delegate trong C# tương tự như khai báo một biến. Nhưng cần thêm từ khóa Delegate để xác định đây là một Delegate. Đồng thời vì Delegate là tham chiếu đến hàm, nên cũng cần khai báo kèm kiểu dữ liệu trả về và tham số đầu vào của Delegate tương ứng với hàm tham chiếu.

Cú pháp:

delegate <kiểu trả về> <tên delegate> (<danh sách tham số nếu có>);

Ví dụ:

delegate int MyDelegate(string s);

Lúc này chúng ta đã tạo một **Delegate** có tên là **MyDelegate**. **MyDelegate** có kiểu trả về là **int**, một tham số đầu vào là **string**.

MyDelegate lúc này có thể dùng làm kiểu dữ liệu cho mọi **Delegate** tới hàm tương ứng kiểu trả về và tham số đầu vào.

1.3. VÍ DỤ DELEGATE

```
public class BasicMaths
{
    public static double Add (double value1, double value2)
    {
        return value1 + value2;
    }

    public static double Sustract (double value1, double value2)
    {
        return value1 - value2;
    }
}

public delegate double MathDelegate (double value1, double value2);
```

Same return type

Same method signature
i.e. Same no of parameters and their types

1.4. CÁC BƯỚC SỬ DỤNG DELEGATE

- ❖ Khai báo delegate
- ❖ Thực hiện delegate tham chiếu tới phương thức
- ❖ Tạo thể hiện của delegate
- ❖ Gọi phương thức thông qua thể hiện delegate

```
// Declare Delegate
public delegate void SampleDelegate(int a, int b);
class MathOperations
{
    public void Add(int a, int b)
    {
        Console.WriteLine("Add Result: {0}", a + b);
    }
    public void Subtract(int x, int y)
    {
        Console.WriteLine("Subtract Result: {0}", x - y);
    }
}
```

```
static void Main(string[] args)
{
    Console.WriteLine("****Delegate Example****");
    MathOperations m = new MathOperations();
    // Instantiate delegate with add method
    SampleDelegate dlgt = m.Add;
    dlgt(10, 90);
    // Instantiate delegate with subtract method
    dlgt = m.Subtract;
    dlgt(10, 90);
    Console.ReadLine();
}
```

****Delegate Example****
Add Result: 100
Subtract Result: -80

1.5. CÁC KIỂU DELEGATE

- **Single Cast Delegates:** Một delegate chỉ tham chiếu đến một phương thức tại một thời điểm

```
SampleDelegate dlgt = m.Add;// thời điểm t1
dlgt(10, 90);

// Instantiate delegate with subtract method

dlgt = m.Subtract;

dlgt(10, 90); // thời điểm t2|
```

1.5. CÁC KIỂU DELEGATE (2)

❑ C# Multicast (đa hướng) Delegates:

- ❖ Có thể tham chiếu đến nhiều phương thức tại cùng một thời điểm
- ❖ Kiểu trả về của multicast delegate phải là kiểu void
- ❖ Dùng toán tử “+” để thêm phương thức vào delegate

```
// Declare Delegate
public delegate void SampleDelegate(int a, int b);
class MathOperations
{
    public void Add(int a, int b)
    {
        Console.WriteLine("Add Result: {0}", a + b);
    }
    public void Subtract(int x, int y)
    {
        Console.WriteLine("Subtract Result: {0}", x - y);
    }
    public void Multiply(int x, int y)
    {
        Console.WriteLine("Multiply Result: {0}", x * y);
    }
}
```

```
static void Main(string[] args)
{
    Console.WriteLine("****Delegate Example****");
    MathOperations m = new MathOperations();
    // Instantiate delegate with add method
    SampleDelegate dlgt = m.Add;
    dlgt += m.Subtract;
    dlgt += m.Multiply;
    dlgt(10, 90);
    Console.ReadLine();
}
```

```
****Delegate Example****
Add Result: 100
Subtract Result: -80
Multiply Result: 900
```

1.5. CÁC KIỂU DELEGATE (3)

❑ C# Multicast (đa hướng) Delegates:

- Khi bạn cần **thực hiện một chuỗi hàm** với cùng kiểu trả về và cùng tham số **đầu vào** mà không muốn gọi nhiều hàm tuần tự (**chỉ gọi 1 hàm 1 lần duy nhất**). Lúc này bạn sẽ cần dùng đến **Multicast Delegate**.
- Bản chất bạn có thể làm một chuỗi **Delegate** cùng kiểu **Delegate** bằng cách dùng toán tử **+**. Lúc này khi bạn gọi **Delegate** sẽ thực hiện tuần tự các **Delegate** được cộng vào với nhau.
- Bạn có thể loại bỏ **Delegate** trong multicast bằng toán tử **-**.

1.6. TRUYỀN THAM SỐ LÀ PHƯƠNG THỨC

- Truyền delegate (tham chiếu đến phương thức) vào phương thức, delegate đóng vai trò tham số.

```
class MathOperations
{
    public void Add(int a, int b)
    {
        Console.WriteLine("Add Result: {0}", a + b);
    }
    public void Subtract(int x, int y)
    {
        Console.WriteLine("Subtract Result: {0}", x - y);
    }
    public void Multiply(int x, int y)
    {
        Console.WriteLine("Multiply Result: {0}", x * y);
    }
}

static void Main(string[] args)
{
    Console.WriteLine("****Delegate Example****");
    MathOperations m = new MathOperations();
    SampleMethod(m.Add, 10, 90);
    SampleMethod(m.Subtract, 10, 90);
    SampleMethod(m.Multiply, 10, 90);
    Console.ReadLine();
}

static void SampleMethod(SampleDelegate dlgt, int a, int b)
{
    dlgt(a, b);
}
```

```
****Delegate Example****
Add Result: 100
Subtract Result: -80
Multiply Result: 900
```

1.7. DÙNG DELEGATE CHO CALL-BACK FUNCTION

Delegate cũng là một biến. Vậy nên mình có thể truyền Delegate vào hàm làm **parameter** như biến bình thường.

Lúc này Delegate này sẽ được gọi là **call-back function**.

Mục đích của việc này là hàm nhận call-back function là param có thể gọi Delegate được đưa vào khi nào cần như ví dụ sau:

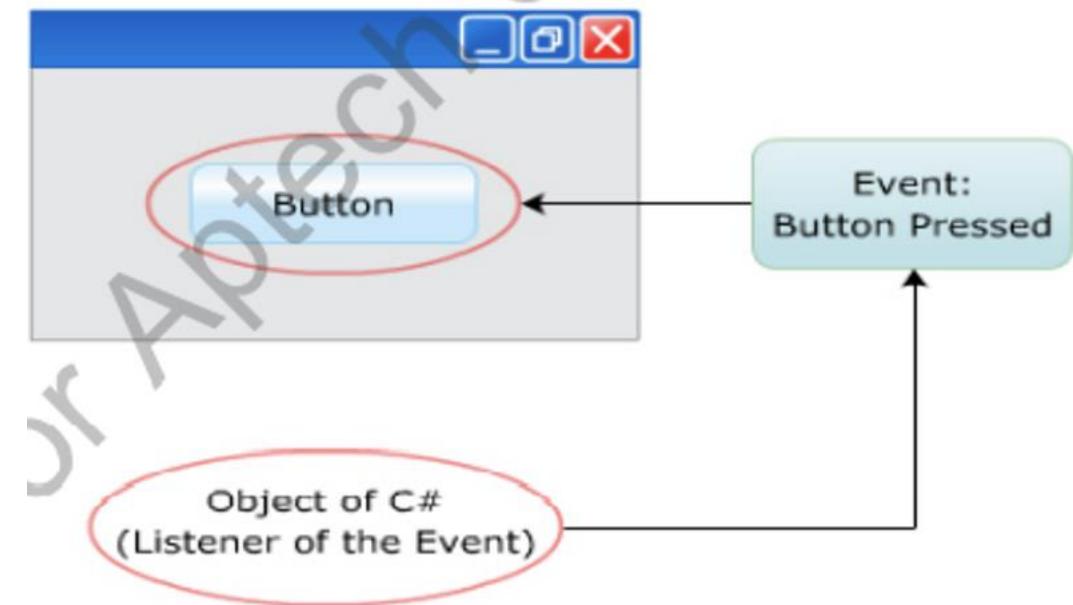
```
delegate int MyDelegate(string s);
static void Main(string[] args)
{
    Console.OutputEncoding = Encoding.Unicode;
    MyDelegate showString = new MyDelegate>ShowString();
    NhapVaShowTen(showString);
    Console.ReadLine();
}

static void NhapVaShowTen(MyDelegate showTen)
{
    Console.WriteLine("Mời nhập tên của bạn:");
    string ten = Console.ReadLine();
    showTen(ten);
}

static int ShowString(string stringValue)
{
    Console.WriteLine(stringValue);
    return 0;
}
```

2.1. EVENT – SỰ KIỆN

- ❑ Sự kiện (Event) là các hành động, ví dụ như nhấn phím, click, di chuyển chuột...
- ❑ Event là một đối tượng đặc biệt của Delegate, nó là nơi chứa các phương thức, các phương thức này sẽ được thực thi khi sự kiện xảy ra
- ❑ Đặc điểm của event:
 - ❖ Được khai báo trong các lớp hoặc interface
 - ❖ Được khai báo là abstract hoặc sealed, virtual
 - ❖ Được thực thi thông qua delegate



2.2. VẤN ĐỀ CỦA DELEGATE VÀ GIẢI QUYẾT CỦA EVENT

Ví dụ dưới đây sẽ dùng **delegate** (đã biết ở phần trước) để xây dựng cơ chế để một lớp này đăng ký nhận sự kiện từ một lớp khác

```
/*
    Publisher là lớp phát đi sự kiện, bằng cách gọi
    một delegate trong phương thức Send
*/
public class Publisher {
    public delegate void NotifyNews (object data);

    public NotifyNews event_news;

    public void Send () {
        event_news?.Invoke ("Co tin moi");
    }
}
```

```
// SubscriberA lớp này đăng ký nhận sự kiện từ Publisher,  
// bằng phương thức Sub, khi sự kiện xảy ra nó sẽ gọi ReceiverFromPublisher  
public class SubscriberA {  
    public void Sub (Publisher p) {  
        p.event_news += ReceiverFromPublisher;  
    }  
  
    void ReceiverFromPublisher (object data) {  
        Console.WriteLine ("SubscriberA: " + data.ToString ());  
    }  
}
```

```
// SubscriberA lớp này đăng ký nhận sự kiện từ Publisher,  
// bằng phương thức Sub - khi đăng ký nó hủy việc nhận sự kiện của các đối tượng khác,  
// khi sự kiện xảy ra nó sẽ gọi ReceiverFromPublisher  
public class SubscriberB {  
    public void Sub (Publisher p) {  
        p.event_news = null; // Hủy các đối tượng khác nhận sự kiện  
        p.event_news += ReceiverFromPublisher;  
    }  
  
    void ReceiverFromPublisher (object data) {  
        Console.WriteLine ("SubscriberB: " + data.ToString ());  
    }  
}
```

```
static void TestDelegate()
{
    Publisher p = new Publisher();
    SubscriberA sa = new SubscriberA();
    SubscriberB sb = new SubscriberB();

    sa.Sub(p);
    sb.Sub(p);

    p.Send();
}
```

SubscriberB: Co tin moi

- Lớp **Publisher** xây dựng một delegate có tên **NotifyNews** và khai báo thuộc tính **event_news** triển khai nó, khi Publisher thi hành **Send()** nó sẽ thi hành delegate này và như vậy những đối tượng nào đăng ký vào **delegate** sẽ có cơ hội nhận thông tin mới từ Publisher
- Hai lớp **SubscriberA** và **SubscriberB** tiến hành đăng ký phương thức **ReceiverFromPublisher** vào Delegate của Pushisher, và như vậy khi chạy code đã có kết quả như trên.

Tuy nhiên, nhìn vào phương thức `public void Sub(Publisher p)` của `SubscriberB` thì đoạn mã:

```
p.event_news = null;
```

Nó đã gán `event_news` bằng `null`, có nghĩa là việc đăng ký của `SubscriberA` lúc trước bị loại bỏ bởi `SubscriberB`, dẫn tới chỉ có `SubscriberB` nhận được tin mới. Điều này là phá hỏng nguyên tắc hoạt động của mô hình lập trình sự kiện - phá vỡ sự đóng gói

Để giải quyết vấn đề trên, thật đơn giản với .NET chỉ cần thêm từ khóa `event` vào định nghĩa `event_news` của Pushliser, và từ đây `event_news` gọi là **Event** chứ không còn gọi là **Delegate**

```
public event NotifyNews event_news;
```

Từ lúc này, các Subscriber chỉ có thể đăng ký nhận sự kiện với toán tử `+=` hoặc hủy nhận sự kiện với toán tử `-=` chứ không thể thực hiện gán `p.event_news = null` vì nếu viết code như vậy lập tức báo lỗi.

2.3. TẠO VÀ SỬ DỤNG EVENT

- ❖ Định nghĩa delegate cho event
- ❖ Tạo event thông qua delegate
- ❖ Đăng ký để lắng nghe và xử lý event
- ❖ Kích hoạt event

2.4. KHAI BÁO EVENT

Cú pháp:

Khai báo delegate:

Bỏ_tù_truy_cập delegate Kiểu_trả_về Tên_delegate (Danh_sách_tham_số);

Khai báo sự kiện:

BỎ_tÙ_truy_cập event Tên_delegate Tên_sự_kiện;

Ví dụ:

```
using System;

public delegate void PrintDetails(); //khai báo delegate trước

class TestEvent
{
    event PrintDetails Print; //rồi khai báo event
}
```

2.5. KÍCH HOẠT EVENT

Cú pháp:

```
Tên_đối_tượng.Tên_sự_kiện(Danh_sách_tham_số);
```

Ví dụ:

```
using System;
public delegate void PrintDetails();
class TestEvent
{
    event PrintDetails Print;
    void Show()
    {
        Console.WriteLine("Hay hien thi toi ra man hinh.");
    }
    static void Main(string[] args)
    {
        TestEvent objTestEvent = new TestEvent();
        objTestEvent.Print += new PrintDetails(objTestEvent.Show);
        objTestEvent.Print(); //câu lệnh kích hoạt sự kiện
    }
}
```

3.1. COLLECTION

- ❑ Collection là lớp hỗ trợ lưu trữ, quản lý và thao tác với các đối tượng linh hoạt.
- ❑ Có thể lưu trữ một tập hợp đối tượng thuộc nhiều kiểu khác nhau.
- ❑ Hỗ trợ rất nhiều phương thức để thao tác với tập hợp như: tìm kiếm, sắp xếp, đảo ngược, . . .
- ❑ Là một mảng có kích thước động:
 - ❖ Không cần khai báo kích thước khi khởi tạo.
 - ❖ Có thể tăng giảm số lượng phần tử trong mảng một cách linh hoạt.

3.2. SYSTEM.COLLECTION NAMESPACE

LỚP	MÔ TẢ
ArrayList	Lớp cho phép lưu trữ và quản lý các phần tử giống mảng. Tuy nhiên, không giống như trong mảng, ta có thể thêm hoặc xoá phần tử một cách linh hoạt và có thể tự điều chỉnh kích cỡ một cách tự động.
HashTable	Lớp lưu trữ dữ liệu dưới dạng cặp Key – Value . Khi đó ta sẽ truy xuất các phần tử trong danh sách này thông qua Key (thay vì thông qua chỉ số phần tử như mảng bình thường).
SortedList	Là sự kết hợp của ArrayList và HashTable . Tức là dữ liệu sẽ lưu dưới dạng Key – Value . Ta có thể truy xuất các phần tử trong danh sách thông qua Key hoặc thông qua chỉ số phần tử. Đặc biệt là các phần tử trong danh sách này luôn được sắp xếp theo giá trị của Key .
Stack	Lớp cho phép lưu trữ và thao tác dữ liệu theo cấu trúc LIFO (Last In First Out).
Queue	Lớp cho phép lưu trữ và thao tác dữ liệu theo cấu trúc FIFO (First In First Out).
BitArray	Lớp cho phép lưu trữ và quản lý một danh sách các bit. Giống mảng các phần tử kiểu bool với true biểu thị cho bit 1 và false biểu thị cho bit 0. Ngoài ra BitArray còn hỗ trợ một số phương thức cho việc tính toán trên bit.

Interface	Mô tả
IEnumerable<T>	Triển khai nó nếu muốn duyệt phần tử bằng <code>foreach</code> , nó định nghĩa phương thức <code>GetEnumerator</code> trả về một enumerator.
ICollection<T>	Giao diện này được triển khai bởi các generic collection. Với nó lấy tổng phần tử bằng thuộc tính <code>Count</code> , copy các phần tử vào mảng bằng <code>CopyTo</code> , thêm bớt phần tử với <code>Add</code> , <code>Remove</code> , <code>Clear</code> .
IList<T>	Giao diện này kế thừa <code>ICollection<T></code> là một danh sách các phần tử truy cập được theo vị trí của nó. Nó có indexer, phương thức để chèn phần tử xóa phần tử <code>Insert</code> <code>RemoveAt</code> .
ISet<T>	Giao diện triển khai bởi các tập hợp
IDictionary< TKey, TValue >	Giao diện để triển khai loại dữ liệu lưu trữ theo cặp key, value.
ILookup< TKey, TValue >	Giao diện để triển khai loại dữ liệu lưu trữ theo cặp key, value. Nhưng cho phép một key có nhiều giá trị
IComparer< TKey, TValue >	Giao diện để triển khai cho phép so sánh để sắp xếp Collection
IEqualityComparer< TKey, TValue >	Giao diện để triển khai cho phép so sánh bằng

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Text;

class Employee : DictionaryBase
{
    public void Add(int id, string name)
    {
        Dictionary.Add(id, name);
    }
    public void OnRemove(int id)
    {
        Console.WriteLine("You are going to delete record
containing ID: " + id);
        Dictionary.Remove(id);
    }
    public void GetDetails()
    {
        IDictionaryEnumerator objEnumerate =
        Dictionary.Get.GetEnumerator();
        while (objEnumerate.MoveNext())
        {
            Console.WriteLine(objEnumerate.Key.ToString() +
"\t\t" +
            objEnumerate.Value);
        }
    }
    static void Main(string[] args)
    {
```

```
Employee objEmployee = new Employee();
objEmployee.Add(102, "John");
objEmployee.Add(105, "James");
objEmployee.Add(106, "Peter");
Console.WriteLine("Original values stored in
Dictionary");
objEmployee.GetDetails();
objEmployee.OnRemove(106);
Console.WriteLine("Modified values stored in
Dictionary");
objEmployee.GetDetails();
}
}
```

3.3. SYSTEM.COLLECTION.GENERIC NAMESPACE

- ❑ Generic trong C# cho phép định nghĩa một hàm, một lớp mà không cần chỉ ra đối số kiểu dữ liệu là gì.
- ❑ Generic cũng là một kiểu dữ liệu trong C# như int, string, bool,... nhưng nó là một kiểu dữ liệu “tự do”, tùy vào mục đích sử dụng mà nó có thể đại diện cho tất cả các kiểu dữ liệu còn lại.
- ❑ Có thể định nghĩa lớp, interface, phương thức, delegate như là kiểu generic

- ❑ Ví dụ: muốn hàm hoán đổi giá trị 2 số nguyên ta sẽ viết như sau:

```
public static void Swap(ref int a, ref int b)
{
    int temp = a;
    a = b;
    b = temp;
}
```

- ❑ Vấn đề khi muốn hoán đổi 2 số float, 2 số double...? → ~~viết thêm các phương thức Swap?~~ → Generic

☐ Sử dụng Generic cho bài toán swap

```
public static void Swap<T>(ref T a, ref T b)
{
    T temp = a;
    a = b;
    b = temp;
}
```

- ☐ Khi gọi **Swap<int>(ref a, ref b)** thì hàm Swap sẽ chạy và thay ký tự **T** thành kiểu dữ liệu int tương ứng

```
int a = 5, b = 7;
double c = 1.2, d = 5.6;
```

```
Swap<int>(ref a, ref b);
Swap<double>(ref c, ref d);
```

3.4. GENERIC CHO LỚP

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("****Generics Example****");
        // Instantiate Generic Class, string is the type argument
        GenericClass<string> gclass = new GenericClass<string>();
        gclass.msg = "Welcome to Tutlane";
        gclass.genericMethod("Suresh Dasari", "Hyderabad");
        Console.ReadLine();
    }
}
```

```
public class GenericClass<T>
{
    public T msg;
    public void genericMethod(T name, T location)
    {
        Console.WriteLine("{0}", msg);
        Console.WriteLine("Name: {0}", name);
        Console.WriteLine("Location: {0}", location);
    }
}
```

3.5. GENERIC CHO DELEGATE

```
static void Main(string[] args)
{
    Console.WriteLine("****Generic Delegate Example****");
    MathOperations m = new MathOperations();
    // Instantiate delegate with add method
    SampleDelegate<int> dlgt = new SampleDelegate<int>(m.Add);
    Console.WriteLine("Addition Result: " + dlgt(10, 90));
    // Instantiate delegate with subtract method
    dlgt = m.Subtract;
    Console.WriteLine("Subtraction Result: " + dlgt(10, 90));
    Console.ReadLine();
}
```

```
// Declare Generic Delegate
public delegate T SampleDelegate<T>(T a, T b);
class MathOperations
{
    public int Add(int a, int b)
    {
        return a + b;
    }
    public int Subtract(int x, int y)
    {
        return x - y;
    }
}
```

3.6. ĐẶC ĐIỂM CỦA GENERIC

- ❖ Giúp tối đa hóa sử dụng lại code (chỉ cần viết 1 hàm có thể tái sử dụng cho nhiều kiểu dữ liệu), an toàn và tăng tốc độ.
- ❖ Có thể dùng generic với: interfaces, classes, methods, events, delegates, structs
- ❖ Có thể tạo generic class với ràng buộc cho các method trong class
- ❖ Có thể lấy thông tin của kiểu dữ liệu được sử dụng bởi generic ở thời điểm runtime bằng Reflection

3.7. LỚP ARRAYLIST

- Là một Collections giúp lưu trữ và quản lý một danh sách các đối tượng theo kiểu mảng (truy cập các phần tử bên trong thông qua chỉ số *index*).
- Rất giống mảng các **object** nhưng có thể thêm hoặc xoá các phần tử một cách linh hoạt và có thể tự điều chỉnh kích cỡ một cách tự động.

Để sử dụng các Collections trong .NET ta cần thêm thư

viện **System.Collections** bằng câu lệnh:

```
using System.Collections;
```

3.7. LỚP ARRAYLIST (2)

Vì ArrayList là một lớp nên trước khi sử dụng ta cần khởi tạo vùng nhớ bằng toán tử **new**:

```
// khởi tạo 1 ArrayList rỗng  
ArrayList MyArray = new ArrayList();
```

Có thể chỉ định sức chứa (Capacity) ngay lúc khởi tạo bằng cách thông qua **constructor** được hỗ trợ sẵn:

```
// khởi tạo 1 ArrayList và chỉ định Capacity ban đầu là 5  
ArrayList MyArray2 = new ArrayList(5);
```

Ngoài ra có thể khởi tạo 1 ArrayList chứa các phần tử được sao chép từ một Collections khác:

```
/*  
 * Khởi tạo 1 ArrayList có kích thước bằng với MyArray2.  
 * Sao chép toàn bộ phần tử trong MyArray2 vào MyArray3.  
 */
```

```
ArrayList MyArray3 = new ArrayList(MyArray2);
```

3.7. LỚP ARRAYLIST (3)

Một số thuộc tính

TÊN THUỘC TÍNH	Ý NGHĨA
Count	Trả về 1 số nguyên là số phần tử hiện có trong ArrayList .
Capacity	Trả về 1 số nguyên cho biết số phần tử mà ArrayList có thể chứa (sức chứa). Nếu số phần tử được thêm vào chạm sức chứa này thì hệ thống sẽ tự động tăng lên. Ngoài ra ta có thể gán 1 sức chứa bất kỳ cho ArrayList .

3.7. LỚP ARRAYLIST (4)

Một số phương thức

TÊN PHƯƠNG THỨC	Ý NGHĨA
Add(object Value)	Thêm đối tượng Value vào cuối ArrayList .
AddRange(ICollection ListObject)	Thêm danh sách phần tử ListObject vào cuối ArrayList .
BinarySearch(object Value)	Tìm kiếm đối tượng Value trong ArrayList theo thuật toán tìm kiếm nhị phân. Nếu tìm thấy sẽ trả về vị trí của phần tử ngược lại trả về giá trị âm. Lưu ý: là ArrayList phải được sắp xếp trước khi sử dụng hàm.
Clear()	Xoá tất cả các phần tử trong ArrayList .
Clone()	Tạo 1 bản sao từ ArrayList hiện tại.
Contains(object Value)	Kiểm tra đối tượng Value có tồn tại trong ArrayList hay không.
GetRange(int StartIndex, int EndIndex)	Trả về 1 ArrayList bao gồm các phần tử từ vị trí StartIndex đến EndIndex trong ArrayList ban đầu.

3.7. LỚP ARRAYLIST (5)

Một số phương thức

IndexOf(object Value)	Trả về vị trí đầu tiên xuất hiện đối tượng Value trong ArrayList . Nếu không tìm thấy sẽ trả về -1.
Insert(int Index, object Value)	Chèn đối tượng Value vào vị trí Index trong ArrayList .
InsertRange(int Index, ICollection ListObject)	Chèn danh sách phần tử ListObject vào vị trí Index trong ArrayList .
LastIndexOf(object Value)	Trả về vị trí xuất hiện cuối cùng của đối tượng Value trong ArrayList . Nếu không tìm thấy sẽ trả về -1.
Remove(object Value)	Xoá đối tượng Value xuất hiện đầu tiên trong ArrayList .
Reverse()	Đảo ngược tất cả phần tử trong ArrayList .
Sort()	Sắp xếp các phần tử trong ArrayList theo thứ tự tăng dần .
ToArray()	Trả về 1 mảng các object chứa các phần tử được sao chép từ ArrayList .

3.7. LỐP ARRAYLIST (6)

```
static void Main(string[] args)
{
    ArrayList a1 = new ArrayList();
```

1 Defining an array list

```
a1.Add(1);
a1.Add("Example");
a1.Add(true);
```

2 Adding elements to the array list

```
Console.WriteLine(a1[0]);
Console.WriteLine(a1[1]);
Console.WriteLine(a1[2]);
```

3 Displaying the elements of the array list


```
static void Main(string[] args)
{
    ArrayList a1 = new ArrayList();
```

Count of items in the Array list

```
a1.Add(1);
a1.Add("Example");
a1.Add(true);
```

1

```
Console.WriteLine(a1.Count);
```

2 Checking to see if the ArrayList contains the element

```
Console.WriteLine(a1.Contains(2));
```

3

```
Console.WriteLine(a1[1]);
a1.RemoveAt(1);
```

Removing an element and showing that the element has been removed

3.8. LỚP HASHTABLE

- ❑ Là một Collections lưu trữ dữ liệu dưới dạng cặp Key – Value
 - ❖ Key đại diện cho 1 khoá giống như chỉ số phần tử của mảng
 - ❖ Value chính là giá trị tương ứng của khoá, dùng Key để truy cập đến Value tương ứng

Array

Value
New York
Boston
Mexico
Kansas
Detroit
California

Hash Table

Key	Value
1	New York
2	Boston
3	Mexico
4	Kansas
5	Detroit
6	California

3.8. LỚP HASHTABLE (2)

- ❑ Vì Key và Value đều là kiểu object nên ta có thể lưu trữ được mọi kiểu dữ liệu từ những kiểu cơ sở đến kiểu phức tạp (class)
- ❑ Do Hashtable là 1 Collections nên để sử dụng ta cần thêm thư viện System.Collections
- ❑ Khai báo:

```
Hashtable MyHash = new Hashtable(); // khởi tạo 1 Hashtable rỗng
```

3.8. LỚP HASHTABLE (3)

❑ Thuộc tính

Property	Description
Count	It is used to get the number of key/value pair elements in hashtable.
IsFixedSize	It is used to get a value to indicate that the hashtable has fixed size or not.
IsReadOnly	It is used to get a value to indicate that the hashtable is readonly or not.
Item	It is used get or set the value associated with the specified key.
IsSynchronized	It is used to get a value to indicate that an access to hashtable is synchronized (thread safe) or not.
Keys	It is used to get the collection of keys in the hashtable.
Values	It is used to get the collection of values in the hashtable.

3.8. LỚP HASHTABLE (4)

❑ Phương thức

Method	Description
Add	It is used to add an element with specified key and value in hashtable.
Clear	It will remove all the elements from hashtable.
Clone	It will create a shallow copy of hashtable.
Contains	It is used determine whether the hashtable contains a specific key or not.
ContainsKey	It is used determine whether the hashtable contains a specific key or not.
ContainsValue	It is used determine whether the hashtable contains a specific value or not.
Remove	It is used to remove an element with specified key from the hashtable.
GetHash	It is used get the hash code for the specified key.

3.8. LỚP HASHTABLE (5)

- Phương thức Add thêm phần tử (key/value) vào hashtable, key phải là duy nhất

Cú pháp:

```
 hashtable_name.Add(key, value);
```

Ví dụ:

```
objTable.Add(001, "John");
objTable.Add(002, "Peter");
objTable.Add(003, "James");
objTable.Add(004, "Joe");
```

```
Hashtable htbl = new Hashtable(){
    {"msg", "Welcome"},  
    {"site", "Tutlane"},  
    {1, 20.5},  
    {2, null}
};
```

3.8. LỚP HASHTABLE (6)

- Phương thức Remove dùng xóa phần tử theo key ra khỏi hashtable

```
static void Main(string[] args)
{
    Hashtable htbl = new Hashtable();
    htbl.Add("msg", "Welcome");
    htbl.Add("site", "Tutlane");
    htbl.Add(1, 20.5f);
    htbl.Add(2, 10);
    htbl.Add(3, 100);
    // Removing hashtable elements with keys
    htbl.Remove(1);
    htbl.Remove("msg");
    Console.WriteLine("*****HashTable Elements*****");
    foreach (DictionaryEntry item in htbl)
    {
        Console.WriteLine("Key = {0}, Value = {1}", item.Key, item.Value);
    }
    Console.ReadLine();
}
```

3.8. LỚP HASHTABLE (7)

- Phương thức Contains(), ContainsKey() and ContainsValue() dùng kiểm tra phần tử có tồn tại?

```
static void Main(string[] args)
{
    Hashtable htbl = new Hashtable();

    htbl.Add("msg", "Welcome");

    htbl.Add("site", "Tutlane");

    htbl.Add(1, 20.5f);

    htbl.Add(2, 10);

    htbl.Add(3, 100);

    Console.WriteLine("Contains Key 4: {0}", htbl.Contains(4));

    Console.WriteLine("Contains Key 2: {0}", htbl.ContainsKey(2));

    Console.WriteLine("Contains Value 'Tutlane': {0}", htbl.ContainsValue("Tutlane"));

    Console.ReadLine();
}
```

3.8. LỚP HASHTABLE (8)

❑ Một số thuộc tính

1. Count

Thuộc tính này dùng để đếm số phần tử thực tế của bảng băm. Ví dụ:

```
if (objTable.Count == 24)  
    Console.WriteLine("Full");
```

2. Keys

Thuộc tính này cung cấp một `ICollection` chứa danh sách các key của bảng băm. Ví dụ:

```
foreach (object key in objTable.Keys)  
    Console.WriteLine(key);
```

3. Values

Thuộc tính này cung cấp một `ICollection` chứa danh sách các value của bảng băm. Ví dụ:

```
foreach (object value in objTable.Values)  
    Console.WriteLine(value);
```

4. IsReadOnly

Thuộc tính này dùng để kiểm tra xem Hashtable có phải là read-only hay không, nếu phải thì trả về `true`, ngược lại trả về `false`.

```
if (objTable.IsReadOnly)  
    Console.WriteLine("Yes");  
else  
    Console.WriteLine("No");
```

3.9. LỚP SORTEDLIST

SortedList lưu trữ các cặp khóa và giá trị theo thứ tự tăng dần của khóa theo mặc định. Lớp SortedList triển khai thực hiện các interface IDictionary và ICollection, do đó các phần tử có thể được truy cập bằng cả khóa và chỉ mục.

C# bao gồm hai loại SortedList là generic và non-generic. Ở đây, chúng ta sẽ tìm hiểu về SortedList non-generic.

3.9. LỚP SORTEDLIST (2)

Thuộc tính	Miêu tả
Capacity	Trả về hoặc thiết lập số lượng phần tử mà đối tượng SortedList có thể lưu trữ.
Count	Trả về số lượng phần tử thực sự có trong SortedList.
IsFixedSize	Trả về một giá trị cho biết liệu SortedList có kích thước cố định hay không.
IsReadOnly	Trả về một giá trị cho biết liệu SortedList là chỉ đọc hay không.
Item	Trả về hoặc thiết lập phần tử tại khóa được chỉ định trong SortedList.
Keys	Trả về danh sách các khóa của SortedList.
Values	Trả về danh sách các giá trị trong SortedList.

3.9. LỚP SORTEDLIST (3)

Phương thức	Miêu tả
Add(object key, object value)	Thêm các cặp khóa và giá trị vào SortedList.
Remove(object key)	Xóa phần tử với khóa được chỉ định.
RemoveAt(int index)	Xóa phần tử tại vị trí được chỉ định.
Contains(object key)	Kiểm tra xem khóa được chỉ định có tồn tại trong SortedList không.
Clear()	Xóa tất cả các phần tử khỏi SortedList.
GetByIndex(int index)	Trả về giá trị theo chỉ mục được lưu trữ trong mảng nội bộ
GetKey(int index)	Trả về khóa được lưu trữ tại chỉ mục được chỉ định trong mảng nội bộ
IndexOfKey(object key)	Trả về một chỉ mục của khóa được chỉ định được lưu trữ trong mảng nội bộ
IndexOfValue(object value)	Trả về một chỉ mục của giá trị được chỉ định được lưu trữ trong mảng nội bộ

3.9. LỚP SORTEDLIST (4)

Thêm các phần tử vào SortedList

Sử dụng phương thức **Add()** để thêm các cặp khóa và giá trị vào SortedList.

Khóa không thể là null và trùng lặp nhưng giá trị có thể là null và trùng lặp. Ngoài ra, kiểu dữ liệu của tất cả các khóa phải giống nhau, để nó có thể so sánh nếu nó sẽ ném ngoại lệ lúc thực thi (runtime).

3.9. LỚP SORTEDLIST (5)

```
SortedList sortedList1 = new SortedList();
sortedList1.Add(3, "Three");
sortedList1.Add(4, "Four");
sortedList1.Add(1, "One");
sortedList1.Add(5, "Five");
sortedList1.Add(2, "Two");
```

```
SortedList sortedList2 = new SortedList();
sortedList2.Add("one", 1);
sortedList2.Add("two", 2);
sortedList2.Add("three", 3);
sortedList2.Add("four", 4);
```

```
SortedList sortedList3 = new SortedList();
sortedList3.Add(1.5, 100);
sortedList3.Add(3.5, 200);
sortedList3.Add(2.4, 300);
sortedList3.Add(2.3, null);
sortedList3.Add(1.1, null);
```

The image shows three separate windows from the Visual Studio debugger, each displaying a `SortedList` instance. Each window has a title bar with the variable name and its count (e.g., `sortedList1 Count = 5`). The lists are presented as tables with two columns: **Keys** and **Values**.

- sortedList1:** Contains 5 items. Keys are 1, 2, 3, 4, 5. Values are "One", "Two", "Three", "Four", "Five".
- sortedList2:** Contains 4 items. Keys are "one", "two", "three", "four". Values are 1, 2, 3, 4.
- sortedList3:** Contains 5 items. Keys are 1.1, 1.5, 2.3, 2.4, 3.5. Values are null, 100, null, 300, 200.

3.9. LỚP SORTEDLIST (6)

Truy cập các phần tử trong SortedList

Có thể truy cập các phần tử của SortedList bằng chỉ mục hoặc khóa. Không giống như các collection khác, SortedList yêu cầu khóa thay vì chỉ mục để truy cập một giá trị cho khóa đó.

Ví dụ sau minh họa truy cập giá trị trong SortedList:

```
1 | SortedList sortedList = new SortedList()
2 | {
3 |     {"one", 1},
4 |     {"two", 2},
5 |     {"three", 3},
6 |     {"four", "Four"}
7 | };
8 |
9 | int i = (int) sortedList["one"];
10 | int j = (int) sortedList["two"];
11 | string str = (string) sortedList["four"];
12 |
13 | Console.WriteLine(i);
14 | Console.WriteLine(j);
15 | Console.WriteLine(str);
```

3.9. LỚP SORTEDLIST (7)

```
SortedList sortedList = new SortedList()
{
    {3, "Three"},
    {4, "Four"},
    {1, "One"},
    {5, "Five"},
    {2, "Two"}
};

int length = sortedList.Count;
for (int i = 0; i < length; i++)
{
    Console.WriteLine("key: {0}, value: {1}",
        sortedList.GetKey(i), sortedList.GetByIndex(i));
}
```

```
key: 1, value: One
key: 2, value: Two
key: 3, value: Three
key: 4, value: Four
key: 5, value: Five
```

3.9. LỚP SORTEDLIST (8)

Xóa các phần tử khỏi SortedList

Sử dụng phương thức **Remove()** hoặc **RemoveAt()** để xóa các phần tử khỏi SortedList.

Ví dụ dưới đây minh họa xóa các phần tử khỏi SortedList:

```
1 | SortedList sortedList = new SortedList();
2 | sortedList.Add("one", 1);
3 | sortedList.Add("two", 2);
4 | sortedList.Add("three", 3);
5 | sortedList.Add("four", 4);
6 |
7 | sortedList.Remove("one");//removes element whose key is 'one'
8 | sortedList.RemoveAt(0);//removes element at zero index i.e first element: four
9 |
10| foreach(DictionaryEntry item in sortedList )
11| {
12|     Console.WriteLine("key: {0}, value: {1}", item.Key, item.Value);
13| }
```

3.9. LỚP SORTEDLIST (9)

Kiểm tra khóa và giá trị tồn tại trong SortedList

Các phương thức `Contains()` và `ContainsKey()` xác định xem khóa được chỉ định có tồn tại trong SortedList hay không.

Phương thức `ContainsValue()` xác định xem giá trị được chỉ định có tồn tại trong SortedList hay không.

3.9. LỐP SORTEDLIST (10)

```
SortedList sortedList = new SortedList();
{
    {3, "Three"},
    {4, "Four"},
    {1, "One"},
    {8, "Eight"},
    {2, "Two"}
};

Console.WriteLine(sList.Contains(2)); // returns true
Console.WriteLine(sList.Contains(4)); // returns true
Console.WriteLine(sList.Contains(6)); // returns false

Console.WriteLine(sList.ContainsKey(2)); // returns true
Console.WriteLine(sList.ContainsKey(6)); // returns false

Console.WriteLine(sList.ContainsValue("One")); // returns true
Console.WriteLine(sList.ContainsValue("Ten")); // returns false
```

True

True

False

True

False

True

False

3.10. LỚP DICTIONARYGENERIC

- Lớp này chứa một tập hợp các phần tử generic được tổ chức thành các cặp key/value, thông qua mỗi key ta sẽ lấy được value tương ứng.
- Không giống như namespace System.Collections, Dictionary generic được dùng để tạo một tập hợp các dữ liệu cùng kiểu và kiểu do người dùng quyết định. Mỗi phần tử ta thêm vào đều sẽ bao gồm một value và một key tương ứng, và ta có thể truy xuất value thông qua key tương ứng đó.
- Lớp Dictionary generic không cho phép chứa phần tử có key là null
- Lớp Dictionary generic thuộc namespace: System.Collections.Generic
- Lớp Dictionary generic có các thuộc tính và phương thức cho phép ta thêm cũng như thao tác với các phần tử trong tập hợp.

3.10. LỚP DICTIONARYGENERIC (2)

Cú pháp khai báo Dictionary generic

```
Dictionary<TKey, TValue> dictionary_name = new Dictionary<TKey, TValue>();
```

Trong đó, TKey và TValue lần lượt là kiểu của key và value sẽ được lưu vào Dictionary generic. Điều này có nghĩa bạn chỉ có thể thêm được các phần tử có key và value tương ứng cùng kiểu đã được chỉ ra.

```
Dictionary<int, string> objDictionary = new Dictionary<int, string>();
```

3.10. LỚP DICTIONARYGENERIC (3)

1. Add()

Thêm một phần tử với key và value cụ thể vào tập hợp. Ví dụ:

[Hide](#) [Copy](#)

```
objDictionary.Add(25, "O dia cung");  
objDictionary.Add(30, "Bo xu ly");  
objDictionary.Add(15, "Bo mach chu");  
objDictionary.Add(65, "Bo nho");
```

2. Remove()

Xoá một phần tử có key tương ứng với đối số của phương thức khỏi dictionary. Ví dụ:

[Hide](#) [Copy](#)

```
objDictionary.Remove(65);
```

3. ContainsKey()

Kiểm tra xem key tương ứng với đối số của phương thức có nằm trong dictionary hay không. Ví dụ:

[Hide](#) [Copy](#)

```
if(objDictionary.ContainsKey(65))
{
    Console.WriteLine("65: " + objDictionary[65]);
}
else
{
    Console.WriteLine("Khong chua");
}
```

4. ContainsValue()

Kiểm tra xem value tương ứng với đối số của phương thức có nằm trong dictionary hay không. Ví dụ:

[Hide](#) [Copy](#)

```
if (objDictionary.ContainsValue("Bo nho"))
{
    Console.WriteLine("Yes");
}
else
{
    Console.WriteLine("No");
}
```

5. GetEnumerator()

Phương thức này sẽ trả về bộ liệt kê interface `IDictionaryEnumerator` thông qua các phần tử của `dictionary` và bạn có thể dùng để liệt kê (iterate) các phần tử trong danh sách. Ví dụ:

Hide **Copy**

```
IDictionaryEnumerator objDictionaryEnumerator = objDictionary.GetEnumerator();
```



6. GetType()

Phương thức này dùng để lấy kiểu của thể hiện hiện thời. Ví dụ:

Hide **Copy**

```
Console.WriteLine(objDictionary.GetType());
```

Kết quả thực thi câu lệnh trên là như sau:

Hide **Copy**

```
System.Collections.Generic.Dictionary`2[System.Int32,System.String]
```



Các thuộc tính của Dictionary generic

1. Count

Thuộc tính này dùng để lấy tổng số cặp key/value trong System.Collections.Generic.Dictionary< TKey, TValue >. Ví dụ:

[Hide](#) [Copy](#)

```
Console.WriteLine(objDictionary.Count);
```

2. Item

Đây là thuộc tính mặc định, thuộc tính này dùng để lấy value hoặc gán value mới cho phần tử có key được chỉ ra. Ví dụ:

[Hide](#) [Copy](#)

```
if (objDictionary.ContainsKey(65))
{
    Console.WriteLine("Gia tri cua key 65: " + objDictionary[65]);
    objDictionary[65] = "Bo luu tru du lieu";
    Console.WriteLine("Sau khi cap nhat, gia tri cua key 65 la: " + objDictionary[65]);
}
```

Trong ví dụ trên, nếu key = 65 (trong cặp ngoặc vuông) tồn tại thì thuộc tính mặc định **Item** sẽ được gọi tới và nó sẽ lấy value tương ứng với key để hiển thị, sau đó **Item** sẽ gán value ứng với key = 65 thành "Bo luu tru du lieu" rồi hiển thị lại.

3. Keys

Cũng giống như Hashtable hay SortedList, thuộc tính **Keys** của Dictionary generic như là mảng một chiều và nó chứa các key của danh sách, **Keys** có thuộc tính Count dùng để lấy số lượng key; **Keys** cũng có một số phương thức riêng của nó như CopyTo() dùng để copy các value tới một mảng khác, phương thức GetType() dùng để lấy kiểu của **Keys**, ... Ví dụ:

[Hide](#) [Copy](#)

```
foreach (object key in objDictionary.Keys)
{
    Console.WriteLine(key);
}
```



4. Values

Thuộc tính **Values** là một mảng đơn chứa các value của danh sách, và **Values** cũng có những thuộc tính và phương thức với ý nghĩa tương tự như thuộc tính **Keys**. Ví dụ:

[Hide](#) [Copy](#)

```
foreach (object value in objDictionary.Values)
{
    Console.WriteLine(value);
}
```



VÍ DỤ 1

```
using System;
using System.Collections;
using System.Collections.Generic;

namespace Demo
{
    class DictionaryCollection
    {
        static void Main(string[] args)
        {
            Dictionary<int, string> objDictionary = new Dictionary<int, string>();
            objDictionary.Add(25, "Hard Disk");
            objDictionary.Add(30, "Processor");
            objDictionary.Add(15, "MotherBoard");
            objDictionary.Add(65, "Memory");
            ICollection objCollect = objDictionary.Keys;
            Console.WriteLine("Original values stored in the collection");
            Console.WriteLine("Keys \t Values");
            Console.WriteLine("-----");
```

```
foreach (int i in objCollect)
{
    Console.WriteLine(i + " \t " + objDictionary[i]);
}
objDictionary.Remove(65);
Console.WriteLine();
if (objDictionary.ContainsKey("Memory"))
{
    Console.WriteLine("Value Memory could not be deleted");
}
else
{
    Console.WriteLine("Value Memory deleted successfully");
}
Console.WriteLine();
Console.WriteLine("Values stored after removing element");
Console.WriteLine("Keys \t Values");
Console.WriteLine("-----");
foreach (int i in objCollect)
{
    Console.WriteLine(i + " \t " + objDictionary[i]);
}
}
```

```
namespace Demo
{
    class Car
    {
        static void Main(string[] args)
        {
            Dictionary<int, string> objDictionary = new
                Dictionary<int, string>();
            objDictionary.Add(201, "Gear Box");
            objDictionary.Add(220, "Oil Filter");
            objDictionary.Add(330, "Engine");
            objDictionary.Add(305, "Radiator");
            objDictionary.Add(303, "Steering");
            Console.WriteLine("Dictionary class contains values of type");
            Console.WriteLine(objDictionary.GetType());
            Console.WriteLine("Keys \t\t Values");
            Console.WriteLine("_____");
            IDictionaryEnumerator objDictionaryEnumerator = objDictionary.GetEnumerator();
            while (objDictionaryEnumerator.MoveNext())
            {
                Console.WriteLine(objDictionaryEnumerator.Key.ToString() + "\t\t" + objDictionaryEnumerator.Value);
            }
        }
    }
}
```

BÀI TẬP

Bài 1: Thực hành lại code sau:

```
public class PrintHelper
{
    // declare delegate
    public delegate void BeforePrint();

    //declare event of type delegate
    public event BeforePrint beforePrintEvent;

    public PrintHelper()
    {

    }

    public void PrintNumber(int num)
    {
        //call delegate method before going to print
        if (beforePrintEvent != null)
            beforePrintEvent();

        Console.WriteLine("Number: {0,-12:N0}", num);
    }

    public void PrintDecimal(int dec)
    {
        if (beforePrintEvent != null)
            beforePrintEvent();

        Console.WriteLine("Decimal: {0:G}", dec);
    }

    public void PrintMoney(int money)
    {
        if (beforePrintEvent != null)
            beforePrintEvent();

        Console.WriteLine("Money: {0:C}", money);
    }
}
```

```
public void PrintTemperature(int num)
{
    if (beforePrintEvent != null)
        beforePrintEvent();

    Console.WriteLine("Temperature: {0,4:N1} F", num);
}
public void PrintHexadecimal(int dec)
{
    if (beforePrintEvent != null)
        beforePrintEvent();

    Console.WriteLine("Hexadecimal: {0:X}", dec);
}
.
```

Viết chương trình tính tổng, hiệu, tích và thương sử dụng cơ chế ủy quyền (delegate).

Bài 2: Kết hợp delegate, event và ArrayList để viết chương trình cho phép nhập và hiển thị nhiều số nguyên dương

Menu chương trình:

```
Add -> Add number  
Display -> Display number  
Exit -> Exit program  
Please choice: ■
```

Khi người dùng nhập “add”: cho phép người dùng nhập vào một số nguyên dương.

Nếu số vừa nhập không thỏa điều kiện, chương trình sẽ hiển thị thông báo “Invalid number.”

```
Add -> Add number  
Display -> Display number  
Exit -> Exit program  
Please choice: add  
Enter number 0  
Invalid number (number > 0)  
Add -> Add number  
Display -> Display number  
Exit -> Exit program  
Please choice:
```

Please try again!” Trường hợp người dùng nhập hợp lệ

```
Add -> Add number  
Display -> Display number  
Exit -> Exit program  
Please choice: add  
Enter number 10  
Add -> Add number  
Display -> Display number  
Exit -> Exit program  
Please choice:
```

Khi người dùng nhập “display”

```
Add -> Add number  
Display -> Display number  
Exit -> Exit program  
Please choice: add  
Enter number 10  
Add -> Add number  
Display -> Display number  
Exit -> Exit program  
Please choice: display  
10  
Add -> Add number  
Display -> Display number  
Exit -> Exit program  
Please choice:
```

Bài 3: Tạo lớp Student có các thuộc tính như sau: studentID, studentName, age, gender và city có phạm vi truy cập là private. Trong lớp này có các phương thức sau: Phương thức nhập chi tiết các thông tin cho Student. Trong phương thức này kiểm tra giá trị nhập vào như sau:

- studentID
- studentName phải có độ dài từ 6-40 ký tự
- age ≥ 18 (chỉ được phép nhập số)
- gender kiểu string và chỉ nhận 1 trong 2 giá trị Nam hoặc Nữ. Ngoài 2 giá trị trên, bắt người dùng phải nhập lại
- city phải có độ dài từ 4-40 ký tự

Chương trình cho phép nhập vào mảng chứa n sinh viên và hiển thị thông tin tất cả sinh viên ra màn hình

Bài 4: Viết chương trình C# minh họa việc sử dụng ArrayList

- Tạo lớp Product với các trường: name, cost, onhand
- Ghi đè phương thức ToString của lớp Product để trả về chuỗi thông tin của Product
- Viết code thêm 5 sản phẩm vào ArrayList và hiển thị ra màn hình

Bài 5: Viết chương trình sử dụng lớp Hashtable theo yêu cầu sau:

- Tạo một hashtable lưu danh sách các ngày trong tuần với key từ 1-8
- Tìm ngày TueDay, in ra thông báo nếu tìm thấy hoặc không
- In ra các ngày trong tuần bao gồm cả key và value

Bài 6: Sử dụng generic viết chương trình hoán vị 2 thành phần có kiểu dữ liệu bất kỳ.

Bài 7: Áp dụng generic cho delegate tham chiếu tới 2 phương thức Add và Subtract của 2 số bất kỳ. Gợi ý:

```
// Declare Generic Delegate
public delegate T SampleDelegate<T>(T a, T b);
class MathOperations
{
    public int Add(int a, int b)
    {
        return a + b;
    }
    public int Subtract(int x, int y)
    {
        return x - y;
    }
}
```

```
W{  
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("****Generic Delegate Example****");
        MathOperations m = new MathOperations();
        // Instantiate delegate with add method
        SampleDelegate<int> dlgt = new SampleDelegate<int>(m.Add);
        Console.WriteLine("Addition Result: " + dlgt(10, 90));
        // Instantiate delegate with subtract method
        dlgt = m.Subtract;
        Console.WriteLine("Subtraction Result: " + dlgt(10, 90));
        Console.ReadLine();
    }
}
```