



# LẬP TRÌNH C# 1

## BÀI 8: COLLECTION & GENERIC

- ◎ Collection
- ◎ Generic



## Phần I: Collection

 Arraylist

 Hashtable

## Phần II: Generic

 Class, method generic

 Delegate generic



- ❑ 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.

## ❑ Collection thông dụng:

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

## □ Nhắc lại Arraylist

```
static void Main(string[] args)
{
```

```
    ArrayList a1 = new ArrayList();
```

1

Defining an array list

Adding elements to the array list

2

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

```
    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();
```

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

1

Count of items in the Array list

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

2

Checking to see if the Array List 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

- ❑ 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

- ❑ 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
```



## □ C# HashTable Properties

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.

## □ C# Hashtable Methods

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 to determine whether the hashtable contains a specific key or not.
ContainsKey	It is used to determine whether the hashtable contains a specific key or not.
ContainsValue	It is used to 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 to get the hash code for the specified key.

- ❑ 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}  
};
```

## ❑ 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();
}
```

- ❑ 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();
}
```

## □ 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");
```



# DEMO

Hiện thực hóa ví dụ của slide trước





# LẬP TRÌNH C# 1

## BÀI 7: COLLECTION & GENERIC (P2)



- ❑ 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);
```

## Generic cho lớp

```
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);
    }
}
```

```
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();
    }
}
```

## Generic cho Delegates

```
// 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;
    }
}
```

```
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();
}
```

## ❑ 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



# DEMO

Hiện thực hóa các ví dụ trong slide



# Tổng kết bài học

## Phần I: Collection

 Arraylist

 Hashtable

## Phần II: Generic

 Class, method generic

 Delegate generic







**KẾT THÚC**