**1. Generic**

**1.1. ý nghĩa**

Ta đã biết về cách dùng ArrayList, với loại Collection này, nó cho phép ta lưu giữ cùng lúc một tập các phần tử có kiểu khác nhau. Ví dụ:

ArrayList list = new ArrayList();

list.Add(1);

list.Add("Aptech");

list.Add(3.45F);

Object x = list[1];

String s = (String)x;

Console.Write("Ket qua: {0}", s);

Rõ ràng, khi ta cần get một phần tử trong list, bắt buộc ta phải chỉ định đúng vị trí phần tử cần lấy, sau đó ta phải nhớ và ép lại kiểu cho nó (từ Object về kiểu nguyên bản) thì ta mới có thể dùng được. Nếu dùng ArrayList theo kiểu này có vẻ hơi “lộn xộn” ta cần một Collection có chức năng như ArrayList nhưng nó chỉ cho phép một loại dữ liệu được lưu giữ trong nó. List là một Collection như vậy, nó sẽ chỉ cho phép một loại dữ liệu được lưu giữ trong nó nếu ta Genneric nó. Ví dụ: List<String> list=new List<String>(); list trong trường hợp này chỉ cho phép dữ liệu kiểu String mới được lưu giữ.

Có thể tổng kết một vài ý nghĩa cơ bản của Generic như sau:

Generic – thống nhất loài, nên nó có thể:

- Đáp ứng nhu cầu lưu giữ dữ liệu thống nhất kiểu (như ví dụ trên);

- Cho phép xây dựng các class, method mà tới khi thực thi ta mới phải chỉ định rõ kiểu dữ liệu được dùng cho các thuộc tính của class hay tham số của method. Việc làm này cho phép class hay method có thể dùng được với nhiều loại dữ liệu khác nhau.

- Ngoài ra với tính “chỉ định một loài” nên Generic còn có mối liên hệ với delegate (các bạn tự tìm tài liệu về vấn đề này).

**1.2. Một số ví dụ**

**1.2.1. Generic method:**

class Program

{

static void Doicho<T>(ref T x, ref T y)//Hàm đổi chỗ đã được Generic (thống nhất loài); từ khóa ref chỉ dấu x, y là tham biến

{

T temp;

temp = x;

x = y;

y = temp;

}

static void Main(string[] args)

{

char c, d;

int a, b;

a = 10;

b = 20;

c = 'I';

d = 'V';

//Hiển thị các giá trị trước khi đổi chỗ:

Console.WriteLine("Các giá trị kiểu int trước khi đổi chỗ:");

Console.WriteLine("a = {0}, b = {1}", a, b);

Console.WriteLine("Các giá trị kiểu char trước khi đổi chỗ:");

Console.WriteLine("c = {0}, d = {1}", c, d);

//Gọi phương thức Doicho

Doicho<int>(ref a, ref b);//Đến lúc này T sẽ được thay thành int và sau đó là char

Doicho<char>(ref c, ref d);

//Như vậy Generic đã thống nhất kiểu dữ liệu cho các đổi số của hàm Doicho thành một loài;

//Nói cách khác nhờ Generic ta có thể sử dụng hàm Doicho này cho nhiều loại dữ liệu khác nhau.

//Hiển thị các giá trị sau khi đổi chỗ:

Console.WriteLine("Các giá trị kiểu int sau khi đổi chỗ:");

Console.WriteLine("a = {0}, b = {1}", a, b);

Console.WriteLine("Các giá trị kiểu int sau khi đổi chỗ:");

Console.WriteLine("c = {0}, d = {1}", c, d);

Console.ReadLine();

}

}

**1.2.2. Generic class:**

class Test<T>

{

T \_value;//Kiểu dữ liệu của \_value được Generic

public Test(T t)

{

this.\_value = t;

}

public void Write()

{

Console.WriteLine(this.\_value);

}

}

class Program

{

static void Main()

{

// Lúc này T được cụ thể hóa bằng int

Test<int> test1 = new Test<int>(5);

test1.Write();

// Lúc này T lại được cụ thể hóa bằng string

Test<string> test2 = new Test<string>("cat");

test2.Write();

Console.ReadLine();

//Như vậy lớp Test có thể dùng được với nhiều kiểu dữ liệu

}

}

**2. Anonymous methods:**

Xét chương trình được viết bằng cách dùng delegate như sau:

class Program

{

public delegate int BD(int x, int y);

public static int Cong(int a, int b)

{

return a + b;

}

static void Main(string[] args)

{

int m = 3, n = 5;

BD k = new BD(Cong);

Console.Write("Ket qua: " + k(m, n));

Console.ReadLine();

}

}

Ta có thể viết lại chương trình trên theo kiểu khác như sau:

class Program

{

public delegate int BD(int x, int y);

//public static int Cong(int a, int b)

//{

// return a + b;

//}

static void Main(string[] args)

{

int m = 3, n = 5;

BD Vietlai = delegate(int a, int b){

return a + b;

};

Console.Write("Ket qua: "+Vietlai(m,n));

Console.ReadLine();

}

}

**Nhận xét:**

Chúng ta thấy phương thức Cong(int a, int b) mà delegate BD đại diện có thể được thay thế bằng một biến có tên “Vietlai” có kiểu BD cùng với một khối (block) lệnh trùng với khối lệnh trong thân của nó. Cách viết lại như vậy cho ta cảm giác không cần cụ thể hóa một hàm do BD đại diện (ở đây là hàm Cong(…)),nhưng thực chất những lệnh bên trong hàm này vẫn được thể hiện nguyên vẹn trong block code của biến Vietlai. Cách làm này người ta gọi là “ẩn phương thức”, thực ra phương thức này nó vẫn tồn tại nhưng dưới một dạng khác mà tên của nó đã được ẩn đi (gọi là Anonymous method là vì vậy).