ĐẠI HỌC XÂY DỰNG HÀ NỘI

BM CNPM – KHOA CNTT

Bài 8

Collections

Collection Generics

Iterator

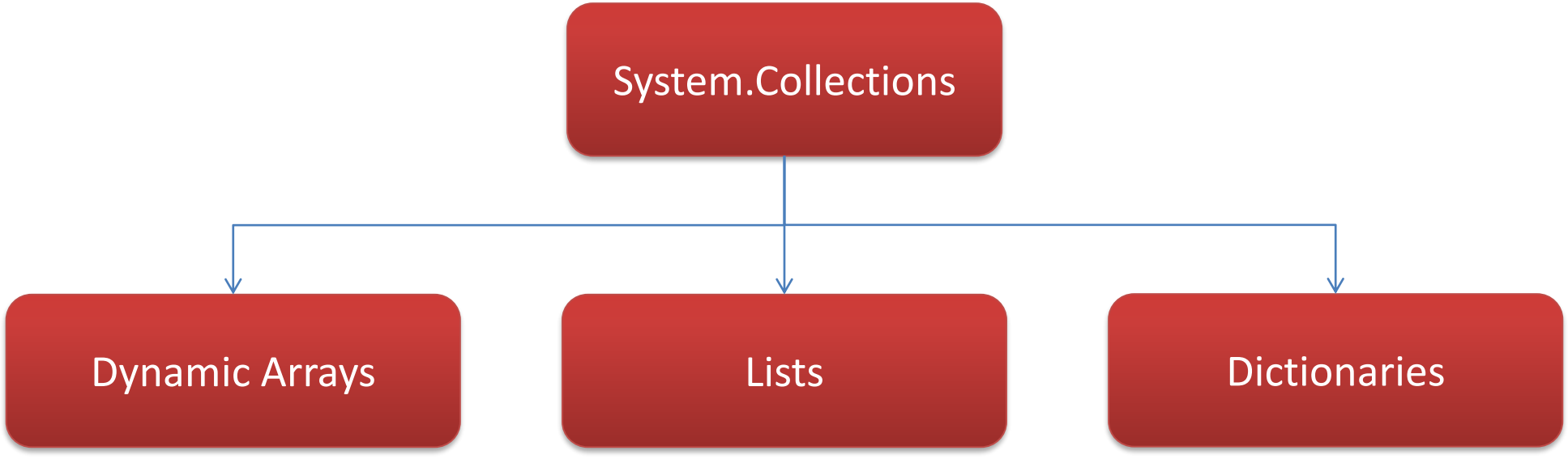
National University of Civil Engineering

Nội dung

- Giới thiệu về tập hợp (Collection)
- Tìm hiểu một số Collection (lớp ArrayList, Hashtable, SortedList)
- Giới thiệu về tập hợp có định kiểu (Generic Collection)
- Tìm hiểu một số Generic Collection (List<>, Dictionary<>, SortedList<>)
- Giới thiệu Iterator
- Cách thực thi Iterator

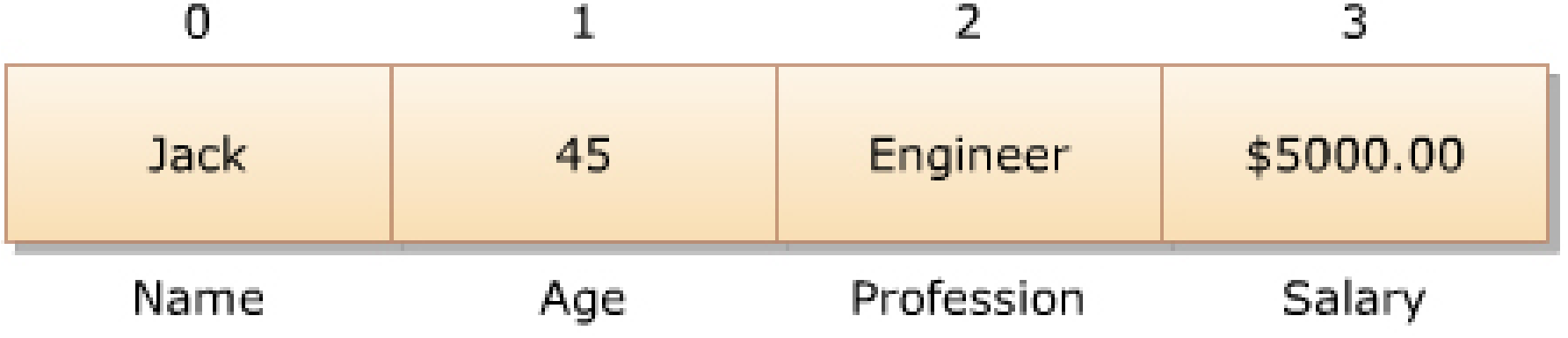
Giới thiệu Collection

- Collection là một tập dữ liệu có liên quan nhưng không nhất thiết phải cùng kiểu. Nó có thể thay đổi động tại thời điểm chạy. Truy cập vào collection giống như truy cập vào mảng.



"ArrayList" class

- ArrayList là mảng động, cho phép lưu trữ các phần tử có giá trị null và trùng nhau
- Giá trị các phần tử có thể thuộc các kiểu khác nhau
- Các phần tử được truy cập thông qua chỉ số (index) và giá trị (value)



Các thao tác trên ArrayList 1-4

Thêm phần tử	
Add	Thêm 1 phần tử vào cuối danh sách
AddRange	Thêm 1 tập phần tử vào cuối danh sách
Insert	Chèn 1 phần tử vào vị trí bất kỳ
InsertRange	Chèn 1 tập phần tử vào vị trí bất kỳ

Ví dụ

```
//tạo mảng
ArrayList arr = new ArrayList();
//thêm giá trị chuỗi
arr.Add("Xin chao");
//thêm giá trị số
arr.Add(10);
//chèn giá trị bool
arr.Insert(1, true);
//tạo mảng names
string[] names = { "Long", "Hai", "Dung" };
//thêm mảng names vào arraylist
arr.AddRange(names);
```

Các thao tác trên ArrayList 2-4

Xóa phần tử	
Remove	Xóa 1 phần tử có giá trị xác đinh
RemoveAt	Xóa 1 phần tử tại vị trí xác định
RemoveRange	Xóa các phần tử tại vị trí xác định
Clear	Xóa tất cả các phần tử

Ví dụ

```
//xóa phần tử có giá trị "Xin chào"
arr.Remove("Xin chào");
//xóa phần tử ở vị trí 1
arr.Remove(1);
//xóa 3 phần tử từ vị trị 1
arr.RemoveRange(1, 3);
//xóa tất cả các phần tử
arr.Clear();
```

- **Duyệt mảng**

Ví dụ

```
//duyệt mảng dùng for
for (int i = 0; i < arr.Count; i++)
{
    Console.WriteLine(arr[i]);
}
//duyệt mảng dùng foreach
foreach (var item in arr)
{
    Console.WriteLine(item);
}
//lấy bộ dữ liệu dạng liệt kê và duyệt dùng while
IEnumerator items = arr.GetEnumerator();
while (items.MoveNext())
{
    Console.WriteLine(items.Current);
}
```

Các thao tác trên ArrayList 4-4

Sắp xếp và tìm kiếm	
Sort	Sắp xếp các phần tử phần biệt chữ hoa chữ thường
IndexOf	Tìm phần tử đầu tiên, nếu tìm thầy trả về vị trí, nếu không trả về -1
LastIndexOf	Tìm phần tử ở cuối (giống IndexOf)
Contains	Kiểm tra trong tập hợp có chứa phần tử cần tìm không

Ví dụ

```
//Tạo mảng
ArrayList tree = new ArrayList() {"Tung", "Cuc","Truc","Mai" };
//sắp xếp phần biệt chữ hoa chữ thường
tree.Sort();
//sắp xếp không phân biệt chữ hoa thường
tree.Sort(new CaseInsensitiveComparer());
//Kiểm tra xem có phần tử "Cuc" không, nếu có thì xóa
if (tree.Contains("Cuc"))
{
    int pos = tree.IndexOf("Cuc");
    tree.RemoveAt(pos);
}
else
    Console.WriteLine("Khong tim thay");
```

Lớp HashTable

- Lớp HashTable là tập hợp để lưu trữ các phần tử, mỗi phần tử gồm một cặp thông tin key(khóa) và value (giá trị)
- Key của các phần tử phải là duy nhất
- Cho phép tìm kiếm phần tử theo key
- **Tạo đối tượng**

  Hashtable pb=new Hashtable();

Các thao tác trên HashTable 1-3

Thêm, xóa phần tử	
Add	Thêm một phần tử vào cuối danh sách
Remove	Xóa một phần tử theo key xác định
Clear	Xóa tất cả các phần tử

Ví dụ

```
//tạo đối tượng
Hashtable pb=new Hashtable();
//thêm 3 phần tử vào hashtable
pb.Add("HR", "Human Resource");
pb.Add("IT", "Information Technology");
pb["MK"] = "Marketing";
//xóa phần tử
pb.Remove("IT");
//xóa hết
pb.Clear();
```

- **Duyệt Hastable và truy xuất tới key, value**

Ví dụ

```
//tạo đối tượng
Hashtable pb=new Hashtable();
//thêm 3 phần tử vào hashtable
pb.Add("HR", "Human Resource");
pb.Add("IT", "Information Technology");
pb["MK"] = "Marketing";
foreach (var key in pb.Keys)
{
    Console.WriteLine(key + ":" + pb[key]);
}
```

Các thao tác trên HashTable 3-3

Tìm kiếm phần tử	
ContainsKey	Trả về true nếu trong tập hợp có chứa key chỉ ra, ngược lại trả về false
ContainsValue	Trả về tru nếu trong tập hợp có chứa value chỉ ra, ngược lại trả về false

Ví dụ

```
//kiểm tra xem trong tập hợp có chứ key là "SC" không, nếu không thì bổ sung vào
if (!pb.ContainsKey("SC"))
    pb.Add("SC","Security");
```

Generic Collection

- Generic là một phần trong hệ thống kiểu của .NET Framework, nó phép định kiểu mà không quan tâm nhiều đến các chi tiết bên trong
- .NET Framework cung cấp nhiều lớp generic trong namespace System.Collections.Generics. Các lớp này hoạt động tương tự các lớp thông thường, tuy nhiên chúng tăng hiệu năng thực hiện và truy xuất an toàn về kiểu (safe-type)
- Ngoài những lớp Generic do .NET cung cấp, người dùng cũng có thể tự tạo ra các lớp Generic tùy biến.

Một số lớp Generic 1-4

- Bảng sau liệt kê danh sách các lớp Collection và Generic Collections tương ứng

Collections	Generic Collections
ArrayList	List<>
Hashtable	Dictionary<>
SortedList	SortedList<>
DictionaryEntry	KeyValuePair<>

Một số lớp Generic 2-4

- Lớp List<>: tương tự ArrayList nhưng các phần tử phải chỉ ra kiểu dữ liệu trước

Ví dụ

```
//tạo tập hợp chỉ chứa kiểu số nguyên
List<int> numbers = new List<int>();
numbers.Add(10);
numbers.Add(3);
numbers.Add(5);
numbers.Add("IT Plus");//dòng này biên dịch sẽ báo lỗi nhé
foreach (int n in numbers)
{
    Console.WriteLine(n);
}
```

Một số lớp Generic 3-4

- Lớp Dictionary<>: tương tự lớp Hashtable nhưng key và value phải được định kiểu trước

Ví dụ

```
Dictionary<int, string> week = new Dictionary<int, string>();
week.Add(2, "Thu 2");
week.Add(3, "Thu 3");
week.Add(4, "Thu 4");
week.Add(5, "Thu 5");
week.Add("6", "Thu 6"); //dòng này sẽ báo lỗi nhé
foreach (int key in week.Keys)
{
    Console.WriteLine(key + ":" + week[key]);
}
```

Một số lớp Generic 4-4

- Lớp StoredList<>: tương tự lớp Dictionary<> nhưng các phần tử được sắp xếp theo key.

Ví dụ

```
SortedList<string, int> number = new SortedList<string, int>();
number.Add("Three", 3);
number["One"] = 1;
number.Add("Two", 2);
//in ra danh sách sắp xếp theo key
foreach (string key in number.Keys)
{
    Console.WriteLine(key + ":" + number[key]);
}
```

Khởi tạo nhanh collection 1-2

- Từ C# 3.0 trở lên, .NET cung cấp cách khởi tạo nhanh một tập hợp mà không cần sử dụng phương thức Add/AddRange như phiên bản trước.

Khởi tạo nhanh với List<>

```
//cách khởi tạo cũ
List<string> animal = new List<string>();
animal.Add("Long");
animal.Add("Ly");
animal.Add("Quy");
animal.Add("Phuong");
//cách khởi tạo mới
List<string> animal1 = new List<string>(){"Long","Ly","Quy","Phuong"};
```

Khởi tạo nhanh collection 2-2

SE - FIT

Khởi tạo nhanh với Dictionary<>

```
//Khởi tạo nhanh Dictionary
Dictionary<string, int> flowers = new Dictionary<string, int>()
{
    {"Hoa Lan",20000},{"Hoa Hong",24000},{"Hoa Ly",23000}
};
```

Khởi tạo nhanh với dữ liệu dạng class

```
//Khởi tạo nhanh với dữ liệu dạng class
List<Employee> list = new List<Employee>()
{
    new Employee{Id=1,Name="Thuy",Address="Ha Noi"},
    new Employee{Id=2,Name="Dung",Address="Ha Noi"},
    new Employee{Id=3,Name="Long",Address="Ha Noi"}
};
```

Iterator

- Một iterator không phải là một thành phần dữ liệu nhưng nó là một cách của việc truy suất đến các phần tử. Nó có thể là một phương thức một khả năng truy xuất get hoặc một một toán tử mà cho phép bạn đánh dấu các giá trị trong một tập hợp. Các Iterator là chỉ định cái cách mà các giá trị được sinh ra khi mà vòng lặp foreach truy xuất đến các phần tử trong một tập hợp. Chúng giữ lại một phần các phần tử trong tập hợp sau đó nó có thể lấy về các giá trị này khi cần đến.

Thực thi Iterator 1-3

- Iterator có thể được thực hiện bằng phương thức GetEnumerator() mà trả về một tham chiếu tới interface IEnumerator.
- Khối iterator sử dụng từ khóa yield. Từ khóa yield return là trả về các giá trị khi đó từ khóa yield break là kết thúc của sử lý iterator.
- Một cách khác để tạo là bằng việc tạo một phương thức kiểu trả về là interface IEnumerable. Đây được gọi là một iterator có tên. Iterator có tên chấp nhận tham số mà được sử dụng cho việc quản lý điểm bắt đầu và điểm kết thúc vủa vòng lặp foreach.

Ưu điểm của Iterator

- Cung cấp một sự đơn giản và nhanh hơn theo cách nhắc lại các giá trị trong tập hợp.
- Giảm bớt sự phức tạp của việc cung cấp một sự liệt kê cho một tập hợp.
- Iterator có thể trả về một lượng lơn các giá trị.
- Iterator có thể được đánh giá và trả về duy nhất các giá trị mà nó cần thiết.
- Iterator có thể trả về các giá trị mà không tốn bộ nhớ bằng việc gọi tới duy nhất một giá trị trong danh sách.

Thực thi Iterator 2-3

Cách 1 thực thi phương thức GetEnumerator của giao diện IEnumberable

```
class Department:IEnumerable
{
    //khai báo mảng dữ liệu
    string[] names = {"Finance","Human Resource", "Information Technology","Marketing" };
    //thực thi phương thức GetEnumberator của giao diện IEnumberable
    public IEnumerator GetEnumerator()
    {
        for (int i = 0; i < names.Length; i++)
        {
            yield return names[i];
        }
    }
    static void Main(string[] args)
    {
        //tạo đối tượng
        Department dep = new Department();
        //sử dung foreach truy xuất tập hợp
        foreach (string item in dep)
        {
            Console.WriteLine(item);
        }
    }
}
```

Thực thi Iterator 3-3

SE - FIT

Cách 2 tạo phương thức có kiểu tra về là IEnumberable

```
class Flower
{
    string[] names = {"Hong","Cuc","Lan","Ly","Mai","Dao" };
    //tạo phương thức có kiểu trả về là IEnumberable
    public IEnumerable GetFlower()
    {
        for (int i = 0; i < names.Length; i++)
        {
            yield return names[i];
        }
    }
    public static void Main(string[] args)
    {
        //tạo đối tượng
        Flower f = new Flower();
        //dùng foreach duyệt qua tập hợp
        foreach (string item in f.GetFlower())
        {
            Console.WriteLine(item);
        }
    }
}
```

Question & Answer