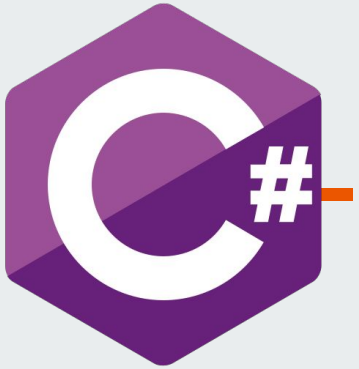


Bài 10

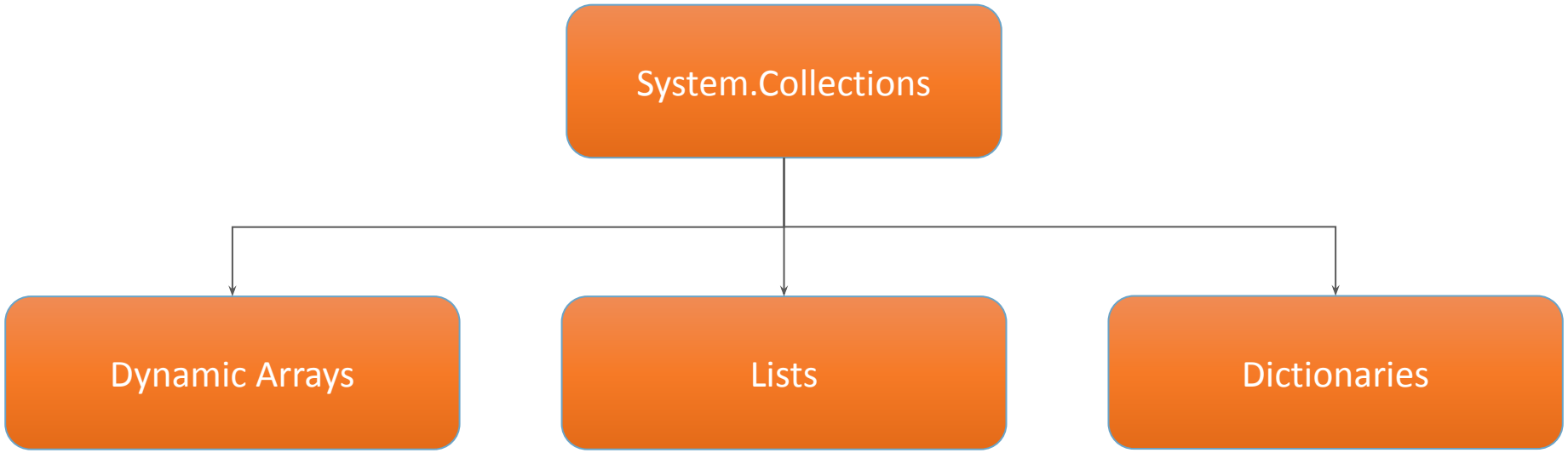
Collections và Iterators

Nội dung

- Giới thiệu về tập hợp (Collection)
- Tìm hiểu một số Collection (lớp ArrayList, Hashtable, 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

```
//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 định
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

```
//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();
```

Các thao tác trên ArrayList 3-4

- Duyệt mảng

```
//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 phân biệt chữ hoa chữ thường
IndexOf	Tìm phần tử đầu tiên, nếu tìm thấy 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

```
//Tạo mảng
ArrayList tree = new ArrayList() { "Tung", "Cuc", "Truc", "Mai" };
//sắp xếp phân 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 1-2

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

```
//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();
```

Các thao tác trên HashTable 2-2

- Duyệt Hashtable và truy xuất tới key, value

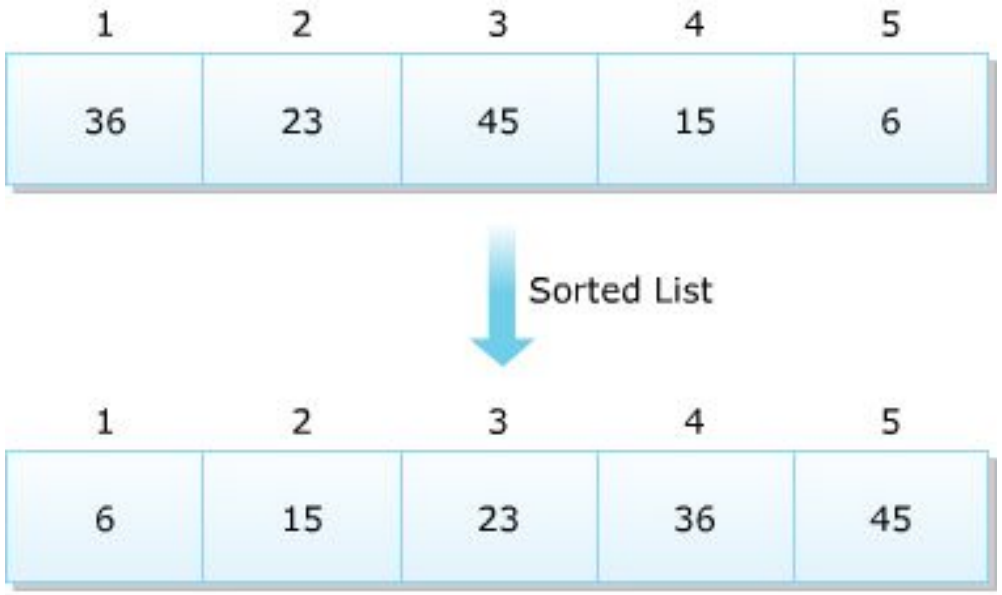
```
//tạo đối tượng
Hashtable pb = new Hashtable();
//thêm 3 phần tử vào hashtable
pb.Add("HR", "Human Resource");
pb.Add("IT", "Information Technology");
pb["MK"] = "Marketing";
foreach (var key in pb.Keys)
{
    Console.WriteLine(key + ":" + pb[key]);
}
```

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 true nếu trong tập hợp có chứa value chỉ ra, ngược lại trả về false

```
//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");
```

Lớp SortedList

- Lớp SortedList giống lớp Hashtable nhưng key được sắp xếp và cho truy xuất phần tử theo chỉ số hoặc theo key.
- Tuy nhiên về tốc độ thì chương trình sử dụng SortedList chậm hơn so với chương trình sử dụng Hashtable



Thao tác với SortedList 1-2

- **Thêm phần tử**

  Sử dụng phương thức **Add()** để thêm một phần tử với (key-value) vào cuối danh sách

- **Xóa phần tử**

  Sử dụng các phương thức **Remove, RemoveAt, Clear**

- **Truy xuất phần tử**

  Phương thức **GetKey(index)** lấy key của phần tử có index

  Phương thức **GetByIndex(index)** lấy value của phần tử có index

Thao tác với SortedList 2-2

```
//tao doi thuong
SortedList staff = new SortedList();
//them gia tri vao staff
staff.Add("Long", "Administrator");
staff.Add("Hung", "Human Resources");
staff.Add("Thuy", "Finance");
staff.Add("Dung", "Marketing");
staff.Add("Thang", "Manager");
Console.WriteLine("Danh sach nhan vien sap xep theo ten:");
//duyet danh sach truy cap theo chi so
for (int i = 0; i < staff.Count; i++)
{
    Console.WriteLine("Key:" + staff.GetKey(i) + "/ Values:" + staff.GetByIndex(i));
}
//tim kiem
if (!staff.ContainsKey("Hoang"))
    staff.Add("Hoang", "Support");
Console.WriteLine("Danh sach nhan vien sap xep theo ten:");
//duyet danh sach truy cap theo key
foreach (var item in staff.Keys)
{
    Console.WriteLine("Key:" + item + "/ Values:" + staff[item]);
}
```

Iterator

- Iterator trong C# được dùng để duyệt qua một danh sách các giá trị của tập hợp. Iterator là một khối mã lệnh trong đó dùng đến vòng lặp foreach để tham chiếu liên tiếp tới tập hợp các giá trị. Iterator sẽ xác định cách mà các giá trị được tạo ra khi vòng lặp foreach truy cập vào các phần tử của tập hợp, và iterator sẽ lưu vết của các phần tử đó, điều này sẽ giúp tăng tốc độ truy cập giá trị của các phần tử.

Thực thi Iterator

- 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 xử 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 của 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 lớn 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 - cách 1

```
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];
        }
    }
}
class Program
{
    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 - cách 2

```
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];
        }
    }
}
class Program
{
    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);
        }
    }
}
```

HỎI ĐÁP

TRẢI NGHIỆM THỰC HÀNH