EASY C#

COLLECTION

Lương Trần Hy Hiến - hyhien@gmail.com

Collections – Tập hợp

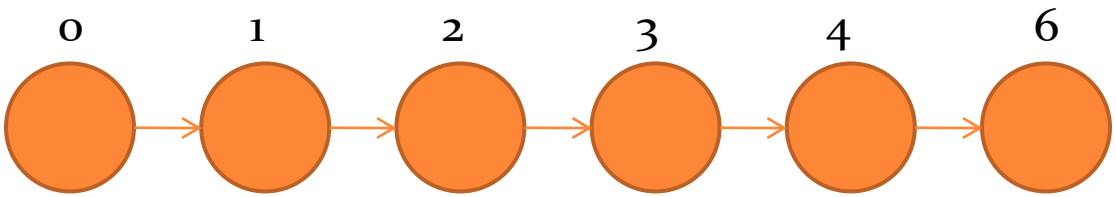
- Là cấu trúc lưu trữ các phần tử có kiểu dữ liệu khác nhau và không hạn chế số lượng phần tử.
- Các lớp có kiểu tập hợp nằm trong namespace **System.Collections** bao gồm:
  - List
  - ArrayList
  - HashTable
  - Queue
  - Stack

NỘI DUNG

- MẢNG ĐỘNG
  - ArrayList
  - List
- TỪ ĐIỂN
  - Hastable
  - Dictionary
- ĐỌC GHI ĐỐI TƯỢNG TỪ FILE
  - Đọc đối tượng từ file
  - Ghi đối tượng từ file

MẢNG ĐỘNG

- Mảng động hay còn gọi là danh sách được sử dụng để quản lý danh sách các phần tử có thứ tự và hỗ trợ các thao tác thêm, xóa, sửa, tìm kiếm,…
- Hình ảnh danh sách như sau



- 2 lớp sau đây được sử dụng để tạo mảng động
  - **List<Kiểu dữ liệu>**: danh sách có kiểu
  - **ArrayList**: danh sách không kiểu

Mảng động – List

- List<kieu_DL> mylist = new List<kieu_DL>();
- Phải chỉ định rõ kiểu dữ liệu khi dùng. VD:
  List<string> mylist = new List<string>();
  - Tạo mảng rỗng chứa số nguyên
    - **List<int> ages = new List<int>();**
  - Tạo mảng số thực khởi đầu 3 phần tử
    - **List<double> salaries = new List<double>()**
      {1.2, 3.4, 5.6};

Mảng động – List

Method / Property	Diễn giải
Add()	Thêm 01 phần tử
Capacity	Tổng số phần tử tối đa
Clear()	Xóa tất cả phần tử
Contains()	Xác định 1 phần tử có trong danh sách hay không?
Count	Số lượng phần tử thật sự
Insert()	Chèn 1 phần tử vào vị trí cụ thể
RemoveAt()	Xóa phần tử tại vị trí
Sort()	Sắp xếp các phần tử

THAO TÁC MẢNG ĐỘNG

- Thao tác thông thường
  - **Add(<kiểu> element)**: thêm phần tử vào mảng
  - **Remove(<kiểu> element)**: xóa phần tử khỏi mảng
  - **[index]**: truy xuất
  - **Count**: lấy số phần tử trong mảng
- Tìm và kiểm tra
  - **Int IndexOf(<kiểu> element)**: tìm vị trí phần tử từ đầu
  - **Int LastIndexOf(<kiểu> element)**: tìm vị trí phần tử từ cuối
  - **Bool Contains(<kiểu>element)**: kiểm tra sự tồn tại
- Các thao tác khác
  - **Clear()**: xóa sạch
  - **Reverse()**: đảo ngược
  - **Sort()**: sắp xếp

VÍ DỤ: MẢNG ĐỘNG - LIST

```
// Tạo mảng rỗng và thêm vào 3 phần tử
List<int> ages = new List<int>();
ages.Add(77);
ages.Add(33);
ages.Add(88);
Console.WriteLine("Số phần tử: {0}", ages.Count);
// Sửa phần tử thứ 2
ages[1] = 55;
// Tìm vị trí phần tử 88
int index = ages.IndexOf(88);
Console.WriteLine("Vị trí của 88: {0}", index);
// Xóa phần tử 77
ages.Remove(77);
// Duyệt và xuất tất cả các phần tử trong mảng
foreach (int age in ages)
{
    Console.WriteLine(" >> Phần tử: {0}", age);
}
// Xóa sạch các phần tử trong mảng
ages.Clear();
```

Thêm các phần tử vào mảng

Mảng động – ArrayList

- Không cần chỉ định kiểu khi khai báo
- Các phần tử có thể có kiểu dữ liệu khác nhau.
  - Tạo mảng rỗng chứa phần tử kiểu bất kỳ
    - **ArrayList items = new ArrayList();**
  - Tạo mảng rỗng có khởi đầu 3 phần tử
    - **ArrayList student = new ArrayList()**
      {“Ngoc Thanh”, true, 80};
    - ArrayList birds = new ArrayList();
      birds.Add(“cat”);
      birds.Add(10952);
      foreach (Bird b in birds)
      b.ToString();

ArrayList Members

- Some Methods:
  - BinarySearch()
  - IndexOf()
  - Sort()
  - ToArray()
  - Remove()
  - RemoveAt()
  - Insert()
- Some Properties:
  - Count
  - Capacity
  - IsFixedSize
  - IsReadOnly
  - IsSynchronized

Lớp Stack<T>

- Mảng động, kiểu tùy ý, kích thước tự động, them/bớt xảy ra tại đỉnh (LIFO)
- Phương thức:
  - `Push(T)` – thêm phần tử vào đỉnh stack
  - `Pop()` – xóa phần tử ở đỉnh stack và trả về giá trị đó

Stack<T> – Ví dụ

- Sử dụng Push(), Pop() và Peek()

```
static void Main()
{
    Stack<string> stack = new Stack<string>();
    stack.Push("1. Ivan");
    stack.Push("2. Nikolay");
    stack.Push("3. Maria");
    stack.Push("4. George");
    Console.WriteLine("Top = {0}", stack.Peek());
    while (stack.Count > 0)
    {
        string personName = stack.Pop();
        Console.WriteLine(personName);
    }
}
```

Lớp Queue<T

- Mảng động, kiểu tùy ý, kích thước động; them/bớt xảy ra 2 chiều (FIFO)
- Phương thức:
  - `Enqueue(T)` – thêm phần tử vào cuối mảng
  - `Dequeue()` – lấy phần tử đầu mảng và trả về giá trị đó

Ví dụ Queue

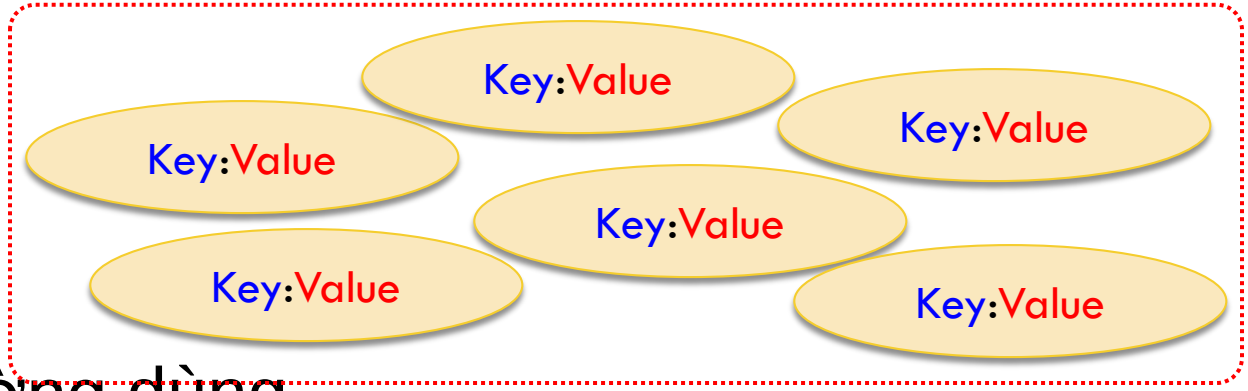

```
static void Main()
{
    Queue<string> queue = new
Queue<string>();
    queue.Enqueue("Message One");
    queue.Enqueue("Message Two");
    queue.Enqueue("Message Three");
    queue.Enqueue("Message Four");
    while (queue.Count > 0)
    {
        string message = queue.Dequeue();
        Console.WriteLine(message);
    }
}
```

THAO TÁC MẢNG KHÔNG KIỂU

- Các hoạt động thao tác mảng không kiểu hoàn toàn tương tự mảng có kiểu chỉ khác duy nhất kiểu của các phần tử thao tác là **object** thay vì **<kiểu>** được chỉ định bởi người dùng.
- Ví dụ:
  - Add(**<kiểu>** element) đối với mảng có kiểu
  - Add(**object** element) đối với mảng không kiểu
- Lưu ý:
  - Khi truy xuất phần tử, bạn cần ép trở lại kiểu của của nó. Ví dụ:
  - **String** HoTen= **(String)**MyArrayList[5];

TỪ ĐIỂN

- Dùng quản lý tập hợp các phần tử không phân biệt thứ tự. Mỗi phần tử gồm 2 phần (khóa và giá trị). Để truy xuất giá trị của một phần tử ta phải biết khóa của nó.
- Hình ảnh



- 2 lớp thường dùng
  - **Dictionary<Kiểu khóa, Kiểu giá trị>:**có kiểu
  - **Hashtable**: không kiểu

KHỞI TẠO TỪ ĐIỂN

```
// Tạo từ điển không kiểu, thêm vào 3 phần tử
Hashtable student = new Hashtable();
student.Add("Name", "Nguyen Van Teo");
student.Add("Age", 30);
student.Add("Gender", true);
// Tạo từ điển có kiểu, thêm 4 phần tử
Dictionary<String, Double> emp = new Dictionary<String, Double>();
emp.Add("Nguyen Thanh Tin", 8.5);
emp.Add("Pham Thi Hoa", 7);
emp.Add("Ngo Quoc Bao", 6.5);
emp.Add("Luong Van Thanh", 9);
// Tạo từ điển có kiểu, khởi đầu 3 phần tử
Dictionary<String, String> words = new Dictionary<String, String>()
{
    {"Love", "Yêu"},
    {"One", "Một"},
    {"School", "Trường học"}
};
```

THAO TÁC TỪ ĐIỂN

- Truy xuất
  - **[Key]**: truy xuất giá trị của phần từ
- Thuộc tính thường dùng
  - **Count**: lấy số phần tử
  - **Keys**: lây tập hợp khóa
  - **Values**: lấy tập hợp giá trị
- Phương thức thường dùng
  - **Add(Key, Value)**: thêm một phần từ
  - **Remove(Key)**: xóa một phần tử
  - **Clear()**: xóa sách
  - **ContainsKey(Key)**: kiểm tra sự tồn tại của khóa
  - **ContainsValue(Value)**: kiểm tra sự tồn tại của giá trị

VÍ DỤ TỪ ĐIỂN

```
// Tạo và khởi dầu từ điển
Dictionary<String, Double[]> marks = new Dictionary<String, Double[]>()
{
    {"Tuấn", new Double[]{7.0, 8.5, 9.5}},
    {"Hoa", new Double[]{5, 7, 4}},
    {"Hồng", new Double[]{6, 8, 10}}
};
// Sửa phần tử thứ hai trong mảng có khóa là "Hoa"
marks["Hoa"][2] = 5.5;
//Duyệt và xuất ra thông tin của mỗi phần tử
foreach (String Name in marks.Keys)
{
    Double[] MA = marks[Name];
    Console.WriteLine("{0}, {1}, {2}, {3}", Name, MA[0], MA[1], MA[2]);
}
Console.WriteLine("Số phần tử: {0}", marks.Count);
```

HashTable

- Là Kiểu từ điển
- Mỗi phần tử bao gồm 01 cặp [key-value]
- Truy xuất nhanh.
- Các cặp key không trùng nhau

Key	Value
masp	SP001
soluong	10
gia	123.45

```
Hashtable ht = new Hashtable();
ht.Add("masp", "SP001");
ht.Add("soluong", 10);
ht.Add("gia", 123.45);
foreach (DictionaryEntry de in ht)
{
    s += string.Format("kqy={0}, value = {1}",
            de.Key, de.Value);
}
```

Bài tập 1

- Nhập danh sách hàng hóa (mã, tên, giá)
- Tìm kiếm hàng hóa theo mã
- Chỉnh sửa hàng hóa tìm được
- Tìm kiếm hàng hóa theo tên (chứa)
- Tìm kiếm hàng hóa theo giá (min, max)
- Hướng dẫn:
  - Dùng 3 danh sách để lưu trữ dữ liệu (List<int>, List<string>, List<double>)

Bài tập 2

- Thực hiện lại bài ở demo1 theo cách sau
  - Sử dụng ArrayList để lưu thông tin hàng hóa (mã, tên, giá)
  - Sử dụng List<ArrayList> để lưu danh sách hàng hóa

Bài tập 3

- Xây dựng một từ điển đơn giản
  - Khởi đầu 10 từ
  - Cho phép tra cứu từ
  - Cho phép nhập thêm từ vào
- Thực hiện lại bài demo 2 theo hướng dẫn
  - Dùng Hastable thay cho ArrayList để lưu thông tin hàng hóa (mã, tên, giá)
  - Sử dụng Dictionary< Hastable> để quản lý hàng hóa