

# Bài: ArrayList trong C#

Xem bài học trên website để ủng hộ Kteam: [ArrayList trong C#](#)

Mọi vấn đề về lỗi website làm ảnh hưởng đến bạn hoặc thắc mắc, mong muốn khóa học mới, nhằm hỗ trợ cải thiện Website. Các bạn vui lòng phản hồi đến Fanpage [How Kteam](#) nhé!

## Dẫn nhập

Ở các bài học trước, chúng ta đã cùng nhau tìm hiểu về [TỔNG QUAN COLLECTIONS TRONG C#](#). Hôm nay chúng ta sẽ cùng tìm hiểu về **ArrayList trong C#**.

## Nội dung

Để đọc hiểu bài này tốt nhất các bạn nên có kiến thức cơ bản về các phần:

- [BIẾN](#), [KIỂU DỮ LIỆU](#), [TOÁN TỬ](#) trong C#
- [CÂU ĐIỀU KIỆN](#) trong C#
- Cấu trúc cơ bản của [VÒNG LẶP](#), [HÀM](#) trong C#
- [MẢNG](#) trong C#
- [LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG TRONG C#](#)

Trong bài học này, chúng ta sẽ cùng tìm hiểu các vấn đề:

- ArrayList là gì?
- Một số thuộc tính và phương thức hỗ trợ sẵn trong ArrayList.

## ArrayList là gì?

**ArrayList** trong C#:

- Là một Collections giúp lưu trữ và quản lý một danh sách các đối tượng theo kiểu mảng (truy cập các phần tử bên trong thông qua chỉ số *index*).
- Rất giống mảng các [object](#) nhưng có thể thêm hoặc xóa các phần tử một cách linh hoạt và có thể tự điều chỉnh kích cỡ một cách tự động.

Để sử dụng các Collections trong **.NET** ta cần thêm thư viện [System.Collections](#) bằng câu lệnh:

```
using System.Collections;
```

Vì ArrayList là một lớp nên trước khi sử dụng ta cần khởi tạo vùng nhớ bằng toán tử **new**:

**C#:**

```
// khởi tạo 1 ArrayList rỗng  
ArrayList MyArray = new ArrayList();
```

Bạn cũng có chỉ định sức chứa (Capacity) ngay lúc khởi tạo bằng cách thông qua **constructor** được hỗ trợ sẵn:

**C#:**

```
// khởi tạo 1 ArrayList và chỉ định Capacity ban đầu là 5  
ArrayList MyArray2 = new ArrayList(5);
```

Ngoài ra bạn cũng có thể khởi tạo 1 ArrayList chứa các phần tử được sao chép từ một Collections khác:

C#:

```

/*
 * Khởi tạo 1 ArrayList có kích thước bằng với MyArray2.
 * Sao chép toàn bộ phần tử trong MyArray2 vào MyArray3.
 */
ArrayList MyArray3 = new ArrayList(MyArray2);

```

## Một số thuộc tính và phương thức hỗ trợ sẵn trong ArrayList

Một số thuộc tính thông dụng trong ArrayList:

TÊN THUỘC TÍNH	Ý NGHĨA
Count	Trả về 1 số nguyên là <b>số phần tử hiện có</b> trong <a href="#">ArrayList</a> .
Capacity	Trả về 1 số nguyên cho biết số phần tử mà <a href="#">ArrayList</a> <b>có thể chứa</b> (sức chứa). Nếu số phần tử được thêm vào chạm sức chứa này thì hệ thống sẽ tự động tăng lên. Ngoài ra ta có thể gán 1 sức chứa bất kỳ cho <a href="#">ArrayList</a> .

Một số phương thức thông dụng trong ArrayList:

TÊN PHƯƠNG THỨC	Ý NGHĨA
Add( <a href="#">object</a> Value)	Thêm đối tượng <b>Value</b> vào cuối <a href="#">ArrayList</a> .
AddRange( <a href="#">ICollection</a> ListObject)	Thêm danh sách phần tử <b>ListObject</b> vào cuối <a href="#">ArrayList</a> .
BinarySearch( <a href="#">object</a> Value)	Tìm kiếm đối tượng <b>Value</b> trong <a href="#">ArrayList</a> theo thuật toán tìm kiếm nhị phân. Nếu tìm thấy sẽ trả về vị trí của phần tử ngược lại trả về giá trị âm. <b>Lưu ý:</b> là ArrayList phải được sắp xếp trước khi sử dụng hàm.
Clear()	Xoá tất cả các phần tử trong <a href="#">ArrayList</a> .
Clone()	Tạo 1 bản sao từ <a href="#">ArrayList</a> hiện tại.
Contains( <a href="#">object</a> Value)	Kiểm tra đối tượng <b>Value</b> <b>có tồn tại</b> trong <a href="#">ArrayList</a> hay không.
GetRange( <a href="#">int</a> StartIndex, <a href="#">int</a> EndIndex)	Trả về 1 ArrayList bao gồm các phần tử từ vị trí <b>StartIndex</b> đến <b>EndIndex</b> trong <a href="#">ArrayList</a> ban đầu.
IndexOf( <a href="#">object</a> Value)	Trả về <b>vị trí đầu tiên</b> xuất hiện đối tượng <b>Value</b> trong <a href="#">ArrayList</a> . Nếu không tìm thấy sẽ trả về <b>-1</b> .
Insert( <a href="#">int</a> Index, <a href="#">object</a> Value)	Chèn đối tượng <b>Value</b> vào vị trí <b>Index</b> trong <a href="#">ArrayList</a> .
InsertRange( <a href="#">int</a> Index, <a href="#">ICollection</a> ListObject)	Chèn danh sách phần tử <b>ListObject</b> vào vị trí <b>Index</b> trong <a href="#">ArrayList</a> .
LastIndexOf( <a href="#">object</a> Value)	Trả về <b>vị trí xuất hiện cuối cùng</b> của đối tượng <b>Value</b> trong <a href="#">ArrayList</a> . Nếu không tìm thấy sẽ trả về <b>-1</b> .
Remove( <a href="#">object</a> Value)	Xoá đối tượng <b>Value</b> <b>xuất hiện đầu tiên</b> trong <a href="#">ArrayList</a> .

Reverse()	Đảo ngược tất cả phần tử trong <a href="#">ArrayList</a> .
Sort()	Sắp xếp các phần tử trong <a href="#">ArrayList</a> theo thứ tự tăng dần.
ToArray()	Trả về 1 mảng các object chứa các phần tử được sao chép từ <a href="#">ArrayList</a> .

Những phương thức trên này cũng khá đơn giản nên mình sẽ không tập trung vào chúng mà để các bạn tự khám phá. Thay vào đó mình sẽ hướng dẫn những thứ hay ho hơn.

Ở trên mình có giới thiệu phương thức **Sort()** sẽ thực hiện sắp xếp danh sách theo thứ tự tăng dần. Vậy nếu danh sách của mình gồm các đối tượng mà mỗi đối tượng là 1 lớp có nhiều thuộc tính thì hàm Sort này biết sắp xếp tăng dần theo thuộc tính nào?

Để trả lời câu hỏi này thì chúng ta cần biết thêm là còn 1 hàm Sort nữa được hỗ trợ sẵn trong [ArrayList](#) có cú pháp như sau:

**Sort** ( [IComparer](#) comparer)

#### Công dụng:

- Hàm này cho phép người dùng tự định nghĩa cách sắp xếp theo ý mình.
- Tham số truyền vào là 1 lớp có kế thừa từ interface [IComparer](#).
- Interface [IComparer](#) chứa 1 phương thức duy nhất là:

**int Comparer** (object x, object y).

Phương thức này sẽ trả về 3 giá trị:

- Bé hơn 0 nếu  $x < y$ .
- Lớn hơn 0 nếu  $x > y$ .
- Bằng 0 nếu  $x = y$ .

Từ những quy định về interface này ta chỉ cần khai báo 1 lớp kế thừa interface [IComparer](#) và định nghĩa nội dung cho phương thức **Comparer** có giá trị trả về theo những quy định trên.

## Ví dụ

Bắt tay vào code nào!

Đầu tiên ta có 1 lớp **Person** đại diện cho 1 con người bao gồm 2 thông tin **Name** và **Age**.

**C#:**

```
public class Person
{
    private string name;
    private int age;

    public string Name
    {
        get { return name; }
        set { name = value; }
    }

    public int Age
    {
        get { return age; }
        set { age = value; }
    }

    /// <summary>
    /// Tạo 1 constructor có tham số để tiện cho việc khởi tạo nhanh đối tượng Person với các giá trị cho sẵn.
    /// </summary>
    /// <param name="Name"></param>
    /// <param name="Age"></param>
    public Person(string Name, int Age)
    {
        this.Name = Name;
        this.Age = Age;
    }

    /// <summary>
    /// Override phương thức ToString để khi cần có thể in thông tin của object ra cho nhanh.
    /// </summary>
    /// <returns></returns>
    public override string ToString()
    {
        return "Name: " + name + " | Age: " + age;
    }
}
```

Tiếp theo ta định nghĩa 1 lớp kế thừa interface `ICompare` và `override` lại phương thức `Compare` trong interface này. Hàm `Compare` sẽ so sánh dựa trên thuộc tính tuổi tăng dần.

**C#:**

```
public class SortPersons : IComparer
{
    public int Compare(object x, object y)
    {
        // Ép kiểu 2 object truyền vào về Person.
        Person p1 = x as Person;
        Person p2 = y as Person;

        /*
         * Vì có thể 2 object truyền vào không phải Person khi đó ta không thể so sánh được.
         * Trường hợp này tốt nhất ta nên ném ra lỗi để lập trình viên sửa chữa.
         * Chi tiết về exception sẽ được trình bày ở những bài học sau.
         */
        if (p1 == null || p2 == null)
        {
            throw new InvalidOperationException();
        }
        else
        {
            /*
             * Khi dữ liệu đã ok thì ta thực hiện so sánh và trả về các giá trị 1 0 -1 tương ứng
             * lớn hơn, bằng, bé hơn.
             */
            if (p1.Age > p2.Age)
            {
                return 1;
            }
            else if (p1.Age == p2.Age)
            {
                return 0;
            }
            else
            {
                return -1;
            }
        }
    }
}
```

Cuối cùng là chương trình chính ta thử tạo 1 [ArrayList](#) và thêm 1 vài đối tượng **Person** vào sau đó gọi hàm **Sort()** và xem kết quả.

**C#:**

```
// Tạo 1 danh sách kiểu ArrayList rỗng
ArrayList arrPersons = new ArrayList();

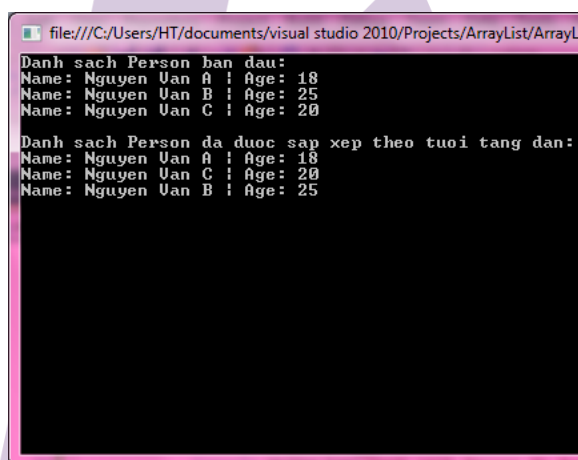
// Thêm 3 Person vào danh sách
arrPersons.Add(new Person("Nguyen Van A", 18));
arrPersons.Add(new Person("Nguyen Van B", 25));
arrPersons.Add(new Person("Nguyen Van C", 20));

// In thử danh sách Person ban đầu ra.
Console.WriteLine("Danh sach Person ban dau: ");
foreach (Person item in arrPersons)
{
    Console.WriteLine(item.ToString());
}

/*
 * Thực hiện sắp xếp danh sách Person theo tiêu chí đã được định nghĩa
 * trong phương thức Compare của lớp SortPerson (tuổi tăng dần).
 */
arrPersons.Sort(new SortPersons());

// In danh sách Person đã được sắp xếp ra màn hình.
Console.WriteLine();
Console.WriteLine("Danh sach Person da duoc sap xep theo tuoi tang dan: ");
foreach (Person item in arrPersons)
{
    Console.WriteLine(item.ToString());
}
```

**Kết quả:** Danh sách đã được sắp xếp theo tuổi tăng dần.



```
file:///C:/Users/HT/documents/visual studio 2010/Projects/ArrayList/ArrayL
Danh sach Person ban dau:
Name: Nguyen Van A : Age: 18
Name: Nguyen Van B : Age: 25
Name: Nguyen Van C : Age: 20
Danh sach Person da duoc sap xep theo tuoi tang dan:
Name: Nguyen Van A : Age: 18
Name: Nguyen Van C : Age: 20
Name: Nguyen Van B : Age: 25
```

Tương tự các hoàn toàn có thể định nghĩa cách so sánh phức tạp hơn như thế nữa...

## Kết luận

Qua bài này chúng ta đã nắm được:

- ArrayList là gì?
- Một số thuộc tính và phương thức hỗ trợ sẵn trong ArrayList.

Bài học sau chúng ta sẽ cùng tìm hiểu về [HASHTABLE TRONG C#](#).

Cảm ơn các bạn đã theo dõi bài viết. Hãy để lại bình luận hoặc góp ý của mình để phát triển bài viết tốt hơn. Đừng quên **"Luyện tập – Thử thách – Không ngại khó"**.

