PHENIKAA UNIVERSITY

Công nghệ .NET

Bài 5 – Làm quen với C# (tiếp)

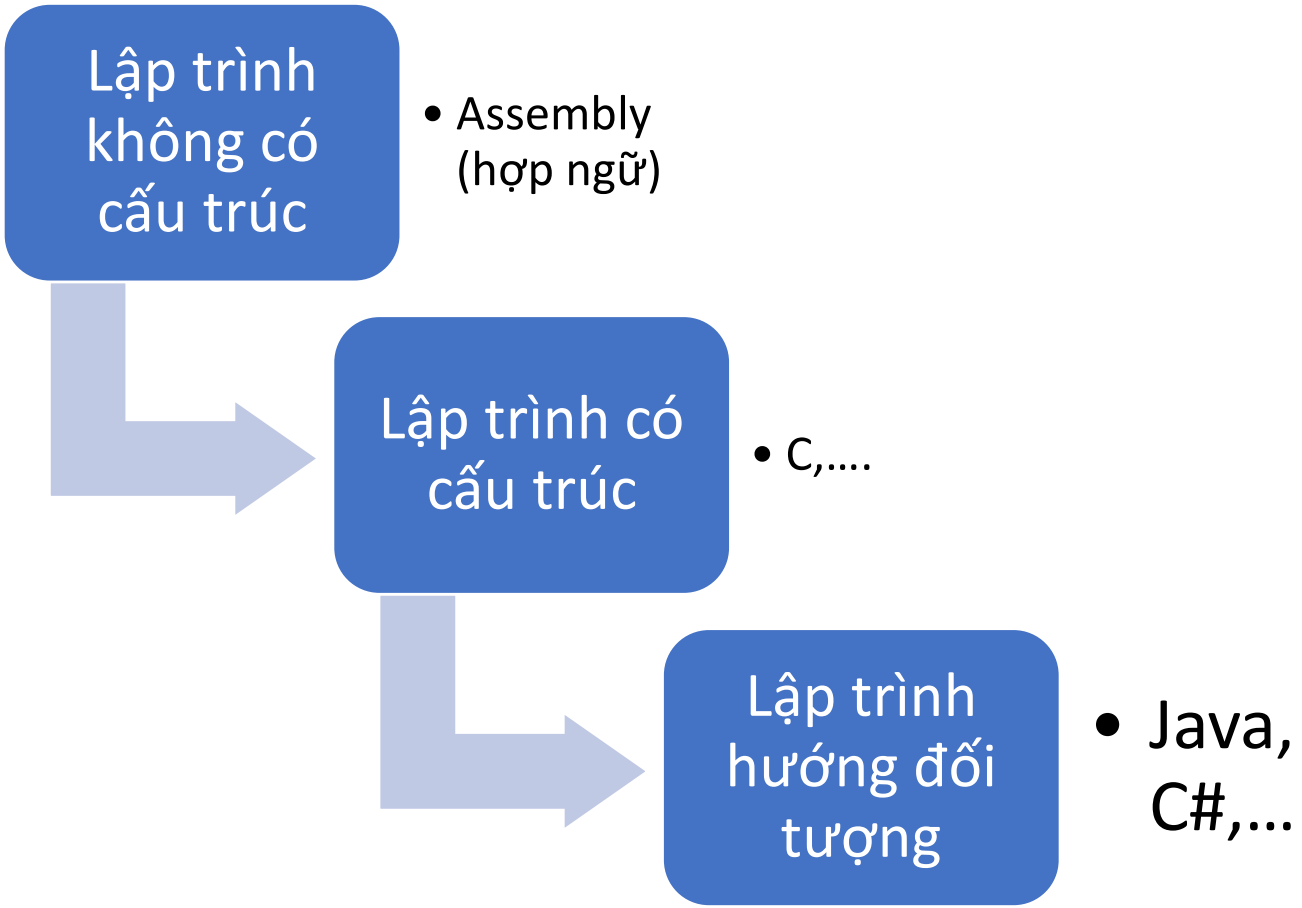
Nguyễn Thành Trung – Khoa CNTT

Email: trung.nguyenthanh1@phenikaa-uni.edu.vn

Nội dung

- Lập trình hướng đối tượng trong C#
- Collection trong C#
  - ArrayList
  - Queue
  - Stack

OOP trong C#


Lập trình không có cấu trúc
• Assembly (hợp ngữ)
Lập trình có cấu trúc
• C,….
Lập trình hướng đối tượng
• Java, C#,…

OOP trong C#

- Khái niệm
  - Đối tượng (Object)
  - Lớp (Class)
- Đặc điểm
  - Tính đóng gói
  - Tính trừu tượng
  - Tính kế thừa
  - Tính đa hình

OOP trong C#

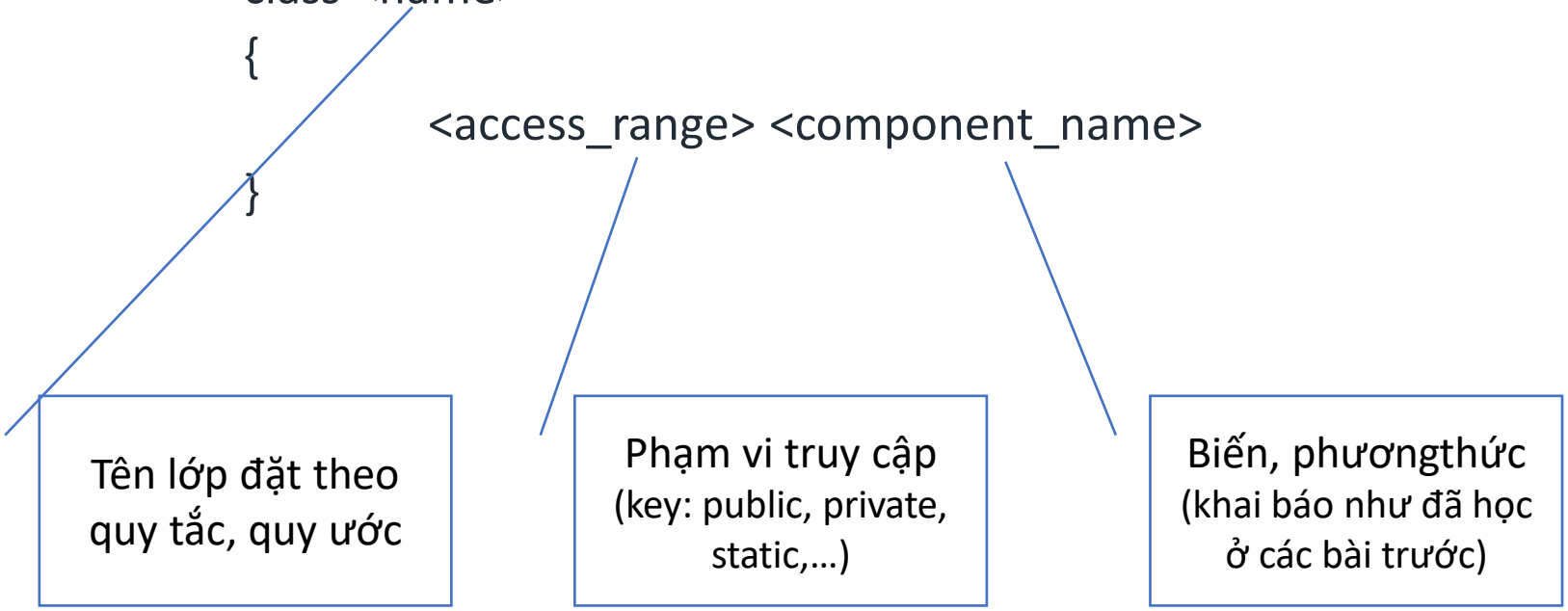
- Class trong C#: gồm các thành phần
  - Thuộc tính: thành phần dữ liệu (biến).
  - Phương thức: hàm thành phần thể hiện các hành vi của một đối tượng thuộc lớp.
  - Phương thức khởi tạo.
  - Phương thức huỷ bỏ.

  => Cũng có thể coi là một kiểu dữ liệu mới do lập trình viên định nghĩa

OOP trong C#

- Class trong C#: gồm các thành phần
  - Khai báo

```
class <name>
{
        <access_range> <component_name>
}
```



OOP trong C#

- Class trong C#: gồm các thành phần
  - Ví dụ

```
class Car
{
        public int Type;
        public string Color;
        public void Start()
        {
                Console.WriteLine("Car is starting");
        }
}
```

  - Khởi tạo: thông qua toán tử new
    - Ví dụ: Car toCross = new Car();

OOP trong C#

- Class trong C#
  - Sử dụng: tương tự struct
    - Ví dụ: toCross.Type = 1; toCross.Start();
  - Phương thức khởi tạo (constructor)
    - Được gọi đến ngay khi khởi tạo một đối tượng
    - Đặc điểm
      - Tên trùng với tên lớp
      - Không có kiểu trả về
      - Nếu không khai báo phương thức khởi tạo, hệ thống sẽ tự tạo phương thức khởi tạo (trống)
    - Có hai loại
      - Không có đối số
      - Có đối số

OOP trong C#

- Class trong C#
  - Phương thức khởi tạo (constructor)
    - Ví dụ

```
class Car
   {
      public int Type;
      public double Weight;
      // không có tham số
      public Car()
      {
         Type = 0;
         Weight = 1500;
      }
      // có tham số
      public Car(int t, double w)
      {
         Type = t;
         Weight = w;
      }
   }
```

OOP trong C#

- Class trong C#
  - Phương thức huỷ bỏ (destructor)
    - Được gọi đến trước khi một đối tượng bị thu hồi
    - Tên trùng với tên lớp nhưng thêm dấu ~
    - Được tự động gọi đến khi một đối tượng thuộc lớp bị thu hồi thông qua cơ chế “rọn rác” tự động (Garbage Collection)

=> Trong C#, GC có cơ chế tự động phát hiện đối tượng không còn được sử dụng nữa để thu hồi, nên không cần thiết phải khai báo tường minh việc huỷ vùng nhớ của nó.

OOP trong C#

- Class trong C#
  - So sánh struct và class

Struct	Class
Kiểu tham trị	Kiểu tham chiếu
Không có destructor	Có destructor
Không có tính kế thừa	Có tính kế thừa

OOP trong C#

- Phạm vi truy cập

Loại	Ý nghĩa
public	Thành phần mang thuộc tính này có thể được truy cập ở mọi nơi
private	Thành phần "riêng tư", chỉ có nội bộ bên trong lớp chứa nó mới có quyền truy cập
protected	Tương tự private + có thể truy cập từ lớp kế thừa lớp đó
internal	Chỉ được truy cập trong cùng một project (khuyến cáo dùng cho class)
protected internal	Tương tự internal + có thể truy cập từ lớp kế thừa lớp đó

OOP trong C#

- Kế thừa trong C#
  - Là cách để một lớp có thể thừa hưởng lại các thuộc tính, phương thức từ một lớp khác.
    - Lớp mới gọi là lớp con
    - Lớp đã có gọi là lớp cha
  - Ví dụ:

```
Class ToyotaCar : Car
{
        public ToyotaCar()
        {
                Weight = 1000;
                Type = 1;
        }
}
```

Không cần phải khai báo lại các thuộc tính

Tuy nhiên, nếu lớp Car khai báo
"private Type" thì sao ?

ArrayList

- Đặc điểm
  - 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 nhưng có thể thêm hoặc xoá các phần tử một cách linh hoạt và có thể tự điều chỉnh kích cỡ một cách tự động.
- Sử dụng
  - using System.Collections;
- Ví dụ:
  - ArrayList listCar = new ArrayList();
  - ArrayList listCar = new ArrayList(10);

ArrayList

- Đặc điểm
  - 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 nhưng có thể thêm hoặc xoá các phần tử một cách linh hoạt và có thể tự điều chỉnh kích cỡ một cách tự động.
- Sử dụng
  - using System.Collections;
- Ví dụ:
  - ArrayList listCar = new ArrayList();
  - ArrayList listCar = new ArrayList(10);

ArrayList

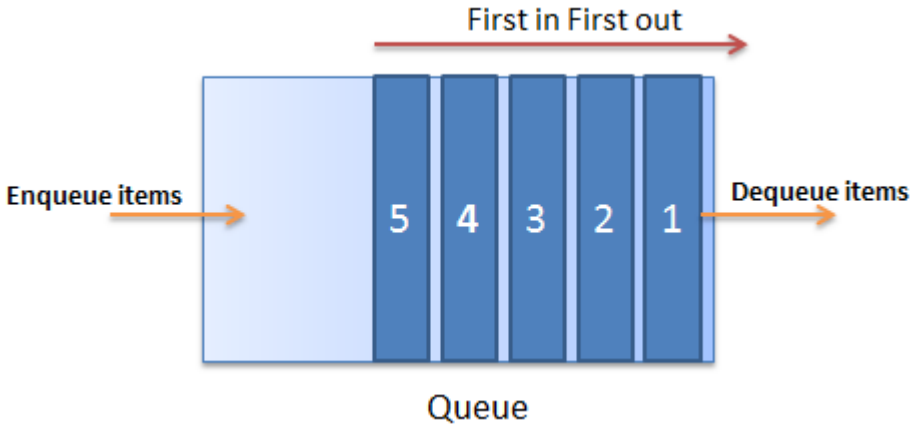
- Một số thuộc tính và phương thức

Tên	Ý nghĩa
Count	Trả về 1 số nguyên là số phần tử hiện có
Capacity	Trả về 1 số nguyên là số phần tử tối đa có thể chưa
Add(object t)	Thêm đối tượng t vào cuối
Clear()	Xoá toàn bộ phần tử
Clone()	Tạo một bảo sao từ ArrayList hiện tại
Contains(object t)	Kiểm tra đối tượng t có tồn tại trong ArrayList không
Insert(int n, object t)	Chèn đối tượng t vào vị trí thứ n trong ArrayList
Remove(object t)	Xoá đối tượng t xuất hiện đầu tiên trong ArrayList
Reverse()	Đảo ngược toàn bộ phần tử
…	

Queue

Queue

- Là một danh sách lưu trữ các đối tượng theo nguyên tắc FIFO
- Hành động thêm phần tử vào Queue được gọi là **Enqueue** (xếp hàng).
- Hành động lấy phần tử ra khỏi Queue được gọi là **Dequeue** (ra khỏi hàng). Và luôn luôn lấy ra phần tử được thêm vào đầu tiên.



Queue

- Sử dụng: using System.Collections;
- Khởi tạo:
  - Queue listCar = new Queue();
  - Queue listCar = new Queue(10);

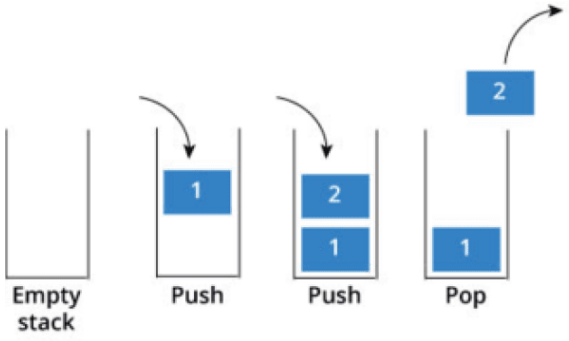
Queue

- Một số thuộc tính và phương thức

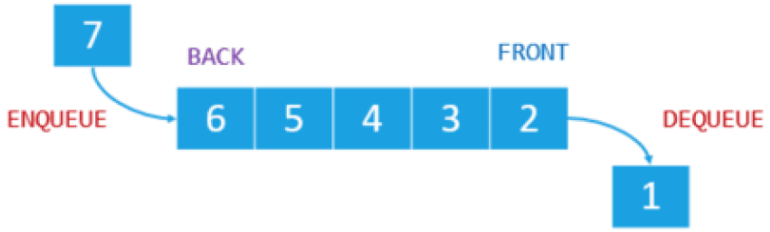
Tên	Ý nghĩa
Count	Trả về 1 số nguyên là số phần tử hiện có
Clear()	Xoá toàn bộ phần tử
Contains(object t)	Kiểm tra đối tượng t có tồn tại trong Queue không
Peek()	Trả về giá trị của đối tượng tại vị trí đầu tiên nhưng không xoá phần tử khỏi Queue
Dequeue()	Trả về giá trị của đối tượng tại vị trí đầu tiên và xoá phần tử khỏi Queue
Enqueue(object t)	Thêm t vào Queue
ToArray()	Trả về 1 mảng các object chứa tất cả các phần tử trong Queue

Stack

- Cách sử dụng tương tự Queue
- Sự khác nhau: LIFO



Stack



Queue

Stack

- Một số thuộc tính và phương thức

Tên	Ý nghĩa
Count	Trả về 1 số nguyên là số phần tử hiện có
Clear()	Xoá toàn bộ phần tử
Contains(object t)	Kiểm tra đối tượng t có tồn tại trong Stack không
Peek()	Trả về giá trị của đối tượng tại vị trí đầu tiên nhưng không xoá phần tử khỏi Stack
Pop()	Trả về giá trị của đối tượng tại vị trí đầu tiên và xoá phần tử khỏi Stack
Push(object t)	Thêm t vào Stack
ToArray()	Trả về 1 mảng các object chứa tất cả các phần tử trong Queue

Stack

- Ngoài ra còn nhiều các Collections hữu ích khác:
  - Multi Threading
  - HashTable
  - SortedList
  - BitArray
  - Generic
  - List
  - Dictionary
  - Tuple
  - Icollection
  - Delegate
  - Event
- Đọc thêm tại:

https://docs.microsoft.com/en-us/dotnet/api/system.collections?view=netframework-4.8

Bài tập

- Viết chương trình:
  - Có các lớp như sau
    - Person: {Name, Age, Gender, Hello()}
    - Teacher: Person + {Class, NotiTeaching()}
    - Student: Person + {Point, InfoStudent()}
  - Tạo các đối tượng Teacher, Student (có thể lưu vào ArrayList, Queue, Stack)
  - Gọi đến các phương thức để thêm object, đếm số phần tử, hiển thị ra màn hình các thông tin đã nhập vào.