|  |  |  |  |
| --- | --- | --- | --- |
| trong .Net | * \n la newline xuống dòng mới * \t: giống như tab trên keyboard. * \r là giống như bạn canh chỉnh column cho thẳng vậy. thẳng trên xuống |  | |
| LinQ | <https://toidicodedao.com/2015/03/26/series-c-hay-ho-linq/> |  | |
| Var | Dùng khai báo biến mà k cần quan tâm đến type(kiểu)  <https://toidicodedao.com/2015/01/22/series-c-hay-ho-nhung-dieu-thu-vi-trong-c-phan-1/> |  | |
| Tim hieu c# | <https://toidicodedao.com/2015/01/22/series-c-hay-ho-nhung-dieu-thu-vi-trong-c-phan-1/> |  | |
| anonymous object | cho phép các loại dữ liệu đóng gói một tập các thuộc tính vào một đối tượng mà không cần phải xác định rõ ràng trước một loại. Đây là một tính năng quan trọng đối với tính năng LINQ giống như SQL được tích hợp vào C # |  | |
|  |  |  | |
| **Hằng trong c#**  (Const) | Hằng là một biến những giá trị k cho thay đổi trong suốt quá trình,  Ngăn chặn việc gán giá trị khác vào | Vd:  Const int X = 10;  Vs việc them const vào tr kiểu dữ liệu thì x sẽ k thay đổi dc giá trị  Lưu ý Khi sừ dụng const thì phải viết hoa toàn bộ tên biến vd : X | |
| **Swich case**  Ý nghĩ là duyệt từ trên xuống dưới và kiểm tra giá trị của biểu thức có = vs giá tri kiểu I dang xét hay k nếu = thì thực hiện câu lệnh I tương ứng | Có 2 dạng :  + dạng thiếu (chỉ có switch và case)  + dạng đủ (có switch , case đồng thời có thêm Default )   * Vs dạng thiếu nếu k thõa bất cứ case nào thì sẽ k làm j còn dạng đủ thì sẽ đi vào Default nếu k thõa tất cả case | Vd: dạng thiếu    Với biểu thức phải trả về kiểu số nguyên hoặc chuổi  Dạng đủ | |
| **Object**  (Kiểu dữ liệu object là kiểu dữ liệu cở bản trong .net mọi kiểu dc kế thừa từ system.object)  **Boxing**  **UnBoxing** | Có 4 phương thức tiêu biểu  + Tostring: trả về string  + GetHashCode : trả về kiểu mã băng của ký tự + Equals : so sánh 2 đối tượng nếu = true khi 2 đối tượng = nhau ngược lại  + Get type : trả về kiểu dữ liệu của đối tượng  Chuyển kiểu dữ liệu giá trị sang kiểu dữ liệu tham chiếu  Ngược lại vs boxing  Hình Minh Họa | Int value =109;  Object objectValue = value;  Ban đầu value có kiểu dữ liệu là int nhưng khi boxing thì giá trị value s3 tham chiếu cho objectValue thành kiểu object objectValue = 109;  Vd: Int value =109;  Object objectValue = value;  Int newValue = (int)objectValue;  Lưu ý kiểm tra kiểu dữ liệu xem có giống nhau hay k nếu k giống thì khi chạy sẽ báo lỗi | |
| **Var**  (là từ khóa hổ trợ khai báo biến mà k cần kiểu dữ liệu, kiểu dữ liệu sẽ dc xác định khi gán giá trị biến lúc đó chương trình sẽ tự ép kiểu về cho biến) | Var phải dc khởi tạo khi dc khai báo và k dc null  Var là từ khóa k phải kiểu dữ liệu thường dc dùng trong duyệt mảng foreach | Vd: var varString = “Quốc Khánh”  Với : var là từ khóa  varString là tên biến | |
| **Kiểu dữ liệu tham chiếu**  **Dynamic** | Từ khóa dynamic là khái niệm mới đưa vào c# 4.0  Các dối tượng thuộc kiểu dynamic sẽ k biết mình thuộc kiểu dữ liệu nào cho đến khi dc thực thi  Sự khác nhau giửa object và dynamic  Phân biệt object , var , dynamic | Vd : khai báo  dynamic tênbiến = “quốc khánh”  Ta khai báo biến stringValue kiểu object  object stringValue = “quốc khánh”  Chúng ta điều biết kiểu chuỗi k hổ trợ toán tử vậy nếu như ta tạo ra (stringValue++;) thì nếu như kiểu object thì sẽ báo lỗi ngay tại stringValue này  Còn nếu là kiểu dynamic thì k báo lỗi nhưng khi chạy sẽ thong báo lỗi | |
| **Goto**  **(có thể tạo ra một vòng lặp)** | Goto dc hiểu là đi điến đâu đó | Vd : khai báo goto  goto CSDL;  CSDL:  Vs CSDL là tên lable , có nghĩa là goto sẽ đi đến biến lable cho dù tr lable có bất kì câu lệnh j cũng đều bỏ qua | |
| **Từ khóa ref và out** | Từ khóa ref  out | Vd : ta tạo 1 biến value kiểu int có giá trị là 5 thì khi ta in ra lần đầu tiên giá trị của nó sẽ = 5 tiếp đến ta cho nó chạy qua 1 hàm Increatvalue (mà trong hàm đó t cho biến value++) thì khi ta in lần 2 thì giá trị nó sẽ k thay đổi vì sao câu trả lời là do trong hàm khi mà cho giá trị nó vào hàm để xử lý thì nó sẽ copy vào ram 1 giá trị y vậy va thực hiện phép tính trong hàm và khi hàm đó kết thúc nó sẽ bị xóa đi nên khi ta in lần 2 giá trị k thay đổi để khác phục đều đó ta cho ref vào trước paramenter    Lưu ý ref k thể gán cho hằng  Còn về out thì khi ta gán out vào trước biến value thì nó sẽ k quan tâm giá trị dc khởi tạo tr là bao nhiêu vì khi ta tạo hàm increasevalue có từ khóa out trong paramenter bắt buộc phải tạo giá trị cho value lúc này ta sẽ in ra màn hình là 5 và 1 | |
| **MẢNG 1 CHIỀU** | Khai báo có những cách sau:  + int [] array = new int [3] ;  + int [] array = new int [] {“phần tử 1”,” phần tử 2”,..};  +int[] array ={1,2,3};  Nhập xuất cơ bản |  | |
| **MẢNG 2 CHIỀU** | CÚ PHÁP:  + String [,] tên mảng = new string [số dòng ,số cột] |  | |
| **Mảng nhiều chiều (3 chiều)** | Cách khai báo thì giống 2 chiều nhưng thay vì 1 dấu phẩy thì mảng 3 chiều xài 2 dấu phẩy |  | |
| Mảng JAGGED |  |  | |
| **Lớp Array trong c#** | <https://www.howkteam.vn/course/khoa-hoc-lap-trinh-c-can-ban/mang-nhieu-chieu-trong-lap-trinh-c-co-ban-1136> |  | |
| **Foreach** | K duyệt mảng hoặc tập hợp chỉ số phần tử giống for  Foreach duyệt tuần tự  Chỉ dùng duyệt mảng và tập ngoài ra k dùng làm j khác | So sánh for vs foreach | |
| **String**  **StringBuilder** | Là một kiểu dữ liệu tham chiếu dùng để lưu trữ chuỗi ký tự  Lớp StringBuilder được .NET xây dựng sẵn giúp chúng ta thao tác trực tiếp với chuỗi gốc và giúp tiết kiệm bộ nhớ hơn so với lớp String.  Đặc điểm của StringBuilder là:   * Cho phép thao tác trực tiếp trên chuỗi ban đầu. * Có khả năng tự mở rộng vùng nhớ khi cần thiết. * Không cho phép lớp khác kế thừa | Một số phương thức lớp string | |
| **Struct** | Khi người ta hỏi struct class thì nói là kiểu dữ liệu có cấu trúc  **Struct** là một kiểu dữ liệu có cấu trúc, được kết hợp từ các kiểu dữ liệu nguyên thuỷ do người lập trình định nghĩa để thuận tiện trong việc quản lý dữ liệu và lập trình.  Là một kiểu dữ liệu tham trị (**kiểu dữ liệu tham trị**đã được trình bày trong bài [KIỂU DỮ LIỆU](https://www.howkteam.vn/redirect?Id=Y%2fHKZv6fbvNJnzXtx4Ic0rZRqPOVzQSxG20E6RGIOYH2ztT5H01j3ihV01MA5mugoAL0u3PcjNdChSHRV0wc0BkVeflYwdzDfL2nyGBnSfGwDQ1dSv2WBaFRyZdAe8e6) )  Dùng để đóng gói các trường dữ liệu khác nhau nhưng có liên quan đến nhau.  Bên trong struct ngoài các biến có kiểu dữ liệu cơ bản còn có các phương thức, các struct khác.  Muốn sử dụng phải khởi tạo cấp phát vùng nhớ cho đối tượng thông qua toán tử new.Struct không được phép kế thừa |  | |
| **Enum** | **Enum**là từ khoá dùng để khai báo một kiểu liệt kê (Enumeration). Kiểu liệt kê là một tập hợp các hằng số do người dùng tự định nghĩa.  Nói cách khác, enum là cách mà C# hỗ trợ người dùng gom nhóm các hằng số lại với nhau và có chung một tên gọi (thường các hằng số này sẽ có liên quan với nhau ví dụ như các trạng thái của 1 sự vật, các tính chất của 1 sự vật, . . .)  Đặc điểm của enum:   * Là một kiểu dữ liệu tham trị (kiểu dữ liệu tham trị đã được trình bày trong bài [KIỂU DỮ LIỆU](https://www.howkteam.vn/redirect?Id=Y%2fHKZv6fbvNJnzXtx4Ic0rZRqPOVzQSxG20E6RGIOYH2ztT5H01j3ihV01MA5mugoAL0u3PcjNdChSHRV0wc0BkVeflYwdzDfL2nyGBnSfGwDQ1dSv2WBaFRyZdAe8e6)) * Enum không được phép kế thừa (khái niệm về kế thừa sẽ trình bày trong bài [KẾ THỪA TRONG C#](https://www.howkteam.vn/redirect?Id=Y%2fHKZv6fbvNJnzXtx4Ic0n3nY4TfYJWT9kAQx46uZ%2f2tQYJMsmQ%2bEzebTh%2bkcLi4ISp4vKw0VlC28mrvwfeB0Uc0WGVxZbqC4w4fuRcirRdM%2fZt3WQp74U3%2f%2bMag3scNnAdOtdkwGQIJH6sKKjW60g%3d%3d)).   Enum k dc kế thừa  Cách nhau = dấu (,) |  | |
| Regular Expression(Regex)(nói cho dễ hiểu nếu muốn cắt chuỗi mà sử dụng cách thong thường thì rất mệt nhưng vs regex thì không) | **Regular Expression** hay tiếng Việt được gọi là Biểu thức chính quy, là một cấu trúc rất mạnh để mô tả một chuỗi theo cách thống nhất chung.  Regular Expression bao gồm tập hợp các ký tự, toán tử hay ký hiệu toán học nhằm biểu thị một chuỗi theo cấu trúc chung mà mọi người học theo. Có thể xem Regular Expression như một loại tiếng lóng dùng chung trong lập trình.  Bạn có thể trích lọc một hay nhiều chuỗi có cấu trúc chung từ một đoạn văn bản hay một chuỗi ra.  Bạn có thể tìm kiếm, thay đổi nội dung của chuỗi một cách dễ dàng. Thay vì phải ngồi cắt chuỗi mỏi mệt như trước đây.  Từ đây, với Regular Expression. Bạn hoàn toàn có thể trích lọc dữ liệu từ các đoạn html theo ý. | Xem ký hiệu và lớp hổ trợ trong regex tại <https://www.howkteam.vn/course/khoa-hoc-lap-trinh-c-can-ban/regular-expression-trong-c-1427> | |
| **Lập trình hướng đối tượng** | | | |
| **Class** | <https://www.howkteam.vn/course/lap-trinh-oop-voi-c/class-trong-lap-trinh-huong-doi-tuong-1370>  (Contructor)hàm dựng là hàm có tên giống với tên class So sánh giữa Struct và Class | | Khởi tạo 1 đối tượng dùng (NEW)  **Class** trong C# chính là cách thể hiện khái niệm về lớp trong lập trình hướng đối tượng.  Một **class**trong C# có các thành phần như:   * Thuộc tính: là các thành phần dữ liệu hay còn gọi là các biến. * Phương thức: là các hàm thành phần thể hiện các hành vi của một đối tượng thuộc lớp. * Phương thức khởi tạo. * Phương thức huỷ bỏ.   **Class**trong C# thực chất là một kiểu dữ liệu mới do người dùng tự định nghĩa.  **Lưu ý: Class**là kiểu dữ liệu tham chiếu vì thế đối tượng dữ liệu thực sự được lưu trên heap.  **+ phương thức khởi tạo** và **phương thức huỷ bỏ** hương thức khởi tạo **Phương thức khởi tạo**(Constructor) là những phương thức đặc biệt được gọi đến ngay khi khởi tạo 1 đối tượng nào đó. Đặc điểm  * Có tên trùng với tên lớp. * Không có kiểu trả về. * Được tự động gọi khi 1 đối tượng thuộc lớp được khởi tạo. * Nếu như bạn không khai báo bất kỳ phương thức khởi tạo nào thì hệ thống sẽ tự tạo ra phương thức khởi tạo mặc định không đối số và không có nội dung gì. * Có thể có nhiều constructor bên trong 1 lớp.   Có 2 loại phương thức khởi tạo:   * Phương thức khởi tạo **không đối số**:   + Là phương thức khởi tạo không có bất kỳ tham số truyền vào nào.   + Thường dùng để khởi tạo các giá trị mặc định cho các thuộc tính bên trong class khi khởi tạo đối tượng (giá trị mặc định này do người lập trình quyết định)   **public** **Cat**()          {              Weight = 800;              Height = 10;       }   * Phương thức khởi tạo **có đối số**:   + Là phương thức khởi tạo có tham số truyền vào. Và khi khởi tạo đối tượng để phương thức này được gọi ta cần truyền đầy đủ các tham số.   + Thường dùng để khởi tạo các giá trị cho các thuộc thuộc tính bên trong class khi khởi tạo đối tượng (các giá trị này do người khởi tạo đối tượng truyền vào).   **public** **Cat**(**int** w, **int** h)          {              Weight = w;              Height = h;     } Lưu ý Khi 1 đối tượng được khởi tạo chỉ có 1 **constructor**phù hợp nhất được gọi cho dù trong lớp có nhiều **constructor**. Phương thức huỷ bỏ **Phương thức huỷ bỏ** (destructor) là phương thức đặc biệt được gọi đến trước khi 1 đối tượng bị thu hồi. Đặc điểm  * Có tên trùng với tên lớp nhưng để phân biệt với constructor thì ta thêm dấu “**~**” vào trước tên lớp. * Không có kiểu trả về. * Được tự động gọi khi 1 đối tượng thuộc lớp kết thúc “vòng đời” của nó thông qua bộ thu dọn rác tự động GC (Garbage Collection). * Nếu bạn không khai báo destructor thì C# sẽ tự động tạo ra 1 destructor mặc định và không có nội dung gì. * Chỉ có 1 destructor duy nhất trong 1 lớp.   Vì bộ GC của C# có cơ chế tự động phát hiện đối tượng không còn được sử dụng nữa và thực hiện thu hồi vùng nhớ của nó nên bạn không cần phải viết tường minh việc huỷ vùng nhớ của nó. Việc bạn có thể làm viết những thứ bạn muốn làm khi đối tượng bị huỷ vào đây thôi! |
| **Các phạm vi truy cập** | Các loại phạm vi truy cập và ý nghĩaPhương thức truy vấn, phương thức cập nhậtTừ khoá get và set (get là trả ra (return))  (set là lấy vào và sử lý) | | **Lưu ý:**   * Nếu khai báo lớp mà không chỉ ra phạm vi cụ thể thì phạm vi mặc định là internal. * Nếu khai báo thành phần bên trong lớp mà không chỉ ra phạm vi cụ thể thì phạm vị mặc định là private.   Trong tính đóng gói có 2 ý chính:   * Các dữ liệu và phương thức có liên quan với nhau được đóng gói thành các lớp để tiện cho việc quản lý và sử dụng. Điều này được thể hiện qua cách ta xây dựng 1 class. * Đóng gói còn để che giấu một số thông tin và chi tiết cài đặt nội bộ để bên ngoài không thể nhìn thấy. Điều này được thể hiện qua các phạm vi truy cập đã trình bày ở trên. Cụ thể:   + **Các thuộc tính** thường sẽ có phạm vi là private. Vì đây chính là các thông tin nội bộ của lớp không thể để truy cập 1 cách tuỳ tiện được (che giấu thông tin).   + **Các phương thức** thường sẽ có phạm vi là public. Vì đây chính là các hành vi (thao tác) mà lớp hỗ trợ cho chúng ta thực hiện những công việc nhất định nên cần phải cho phép mọi người có thể sử dụng được.   Một vấn đề đặt ra là nếu như mọi thuộc tính bên trong lớp đều là private thì khi người dùng bên ngoài muốn xem hoặc thay đổi giá trị thì phải làm sao?  Đến đây đòi hỏi người lập trình phải viết ra **1 cách nào đó** để hỗ trợ người dùng làm điều này. Và cách để người dùng có thể xem hoặc thay đổi giá trị cho các thuộc tính bên trong lớp đó chính là thông qua các **phương thức truy vấn**, **phương thức cập nhật**.   * **Phương thức truy vấn** là phương thức giúp người dùng có thể xem được dữ liệu của 1 thuộc tính nào đó. Cụ thể, phương thức truy vấn chỉ cần trả về giá trị của thuộc tính tương ứng là đủ. * **Phương thức cập nhật** là phương thức giúp người dùng có thể thay đổi giá trị cho 1 thuộc tính nào đó. Cụ thể, phương thức cập nhật chỉ cần thực hiện cập nhật giá trị mới cho thuộc tính tương ứng (có thể kiểm tra tính đúng đắn của dữ liệu trước khi truyền vào).   Một số quy ước nhỏ về cách đặt tên các phương thức này:   * Những phương thức truy vấn nên bắt đầu bằng từ khoá **get** và kèm theo sau là tên thuộc tính tương ứng. Ví dụ: getHoTen(), getDiemToan(), . . . * Những phương thức cập nhật nên bắt đầu bằng từ khoá **set** và kèm theo sau là tên thuộc tính tương ứng. Ví dụ: setDiemToan(), setHoTen(), . . . * Nếu thuộc tính kiểu luận lý (bool) thì tên phương thức truy vấn nên bắt đầu bằng từ khoá **is** và kèm theo sau là tên thuộc tính tương ứng. * Phương thức truy vấn sẽ có kiểu trả về trùng với kiểu dữ liệu của thuộc tính tương ứng và không có tham số truyền vào. * Phương thức cập nhật sẽ có kiểu trả về là void và có 1 tham số truyền vào có kiểu dữ liệu trùng với kiểu dữ liệu của thuộc tính tương ứng.   Trong C#, **phương thức truy xuất**và **phương thức cập nhật** đã được nâng cấp lên thành 1 cấu trúc mới ngắn gọn hơn và tiện dụng hơn đó là **property**.  Sử dụng property giúp ta có thể thao tác dữ liệu tự nhiên hơn nhưng vẫn đảm bảo tính đóng gói của lập trình hướng đối tượng.  Trong đó:   * **<kiểu dữ liệu>**là kiểu dữ liệu của property. Thường sẽ trùng với kiểu dữ liệu của thuộc tính **private**tương ứng bên trong lớp. * **get**, **set**, **value**là từ khoá có ý nghĩa:   + Từ khoá **get**tương đương với phương thức truy vấn. (return)   + Từ khoá **set**tương đương với phương thức cập nhật. (caigido = value)   + Từ khoá **value**đại diện cho giá trị mà người gán vào property (tương đương với tham số truyền vào của phương thức cập nhật). * **<tên thuộc tính>** là tên thuộc tính thực sự bên trong lớp.  Lưu ý:  * Người ta dùng **Property**thay cho phương thức truy vấn, phương thức cập nhật vì thế tên **property**thường phải làm gợi nhớ đến tên thuộc tính **private**bên trong lớp. * Tuỳ theo nhu cầu và tính bảo mật mà người lập trình có thể ngăn không cho gán giá trị hoặc ngăn không cho lấy dữ liệu bằng cách bỏ đi từ khoá tương ứng. |
| **Static(tĩnh)** | Đặc điểm của thành viên tĩnhLớp tĩnh có các đặc điểmĐặc điểm của constructor tĩnh | | Bạn có thể hiểu **biến tĩnh** là:   * Một biến dùng chung cho mọi đối tượng thuộc lớp. * Nó được khởi tạo vùng nhớ 1 lần duy nhất ngay khi chương trình được nạp vào bộ nhớ để thực thi và huỷ khi kết thúc chương trình.   Bình thường các thuộc tính, phương thức sẽ có đặc điểm:   * Chỉ có thể sử dụng sau khi khởi tạo đối tượng. * Dữ liệu thuộc về riêng mỗi đối tượng (xét cùng 1 thuộc tính thì các đối tượng khác nhau thì thuộc tính đó sẽ mang các giá trị khác nhau). * Được gọi thông qua tên của đối tượng.   Đôi lúc người lập trình mong muốn 1 thuộc tính nào đó được dùng chung cho mọi đối tượng (chỉ được cấp phát 1 vùng nhớ duy nhất). Từ đó khái niệm **thành viên tĩnh** ra đời.  Đặc điểm của thành viên tĩnh:   * Được khởi tạo **1 lần duy nhất**ngay khi biên dịch chương trình. * Có thể **dùng chung** cho mọi đối tượng. * **Được gọi thông qua tên lớp**. * Được **huỷ khi kết thúc** chương trình.   Có 4 loại thành viên tĩnh chính:   * Biến tĩnh (static variable). * Phương thức tĩnh (static method). * Lớp tĩnh (static class). * Phương thức khởi tạo tĩnh (static constructor).   Để khai báo 1 **thành viên tĩnh** ta sử dụng từ khoá **static**đặt trước tên biên, tên phương thức hoặc tên lớp. Chi tiết sẽ được trình bày ngày sau đây.  Về cách sử dụng thì bạn thao tác hoàn toàn giống 1 biến bình thường chỉ cần lưu ý là phải gọi biến này thông qua tên lớp.  Chỉ chứa các thành phần tĩnh (biến tĩnh, phương thức tĩnh).  **Không**thể khai báo, khởi tạo 1 đối tượng thuộc lớp tĩnh.  Với 2 đặc điểm trên có thể thấy lớp tĩnh thường được dùng với mục đích khai báo 1 lớp tiện ích chứa các hàm tiện ích hoặc hằng số vì:   * Ràng buộc các thành phần bên trong lớp phải là **static**. * Không cho phép tạo ra các đối tượng dư thừa làm lãng phí bộ nhớ. * Mọi thứ đều được truy cập thông qua tên lớp.   Trong C# có rất nhiều lớp tiện ích sử dụng lớp tĩnh, phương thức tĩnh để khai báo. Ví dụ điển hình đó là lớp **Math**.  Lớp **Math**chứa:   * Các hằng số nhứ **PI**, **E**. * Các phương thức hỗ trợ tính toán như: **sin**, **cos**, **tan**, **sqrt**, **exp**, . . .   **Không được phép** khai báo phạm vi truy cập. Nếu cố tình làm điều này C# sẽ báo lỗi khi biên dịch.  Constructor tĩnh sẽ **được gọi 1 lần duy nhất** khi chương trình được nạp vào bộ nhớ để thực thi như là 1 cách để ta thiết lập một số thông số theo ý muốn trước khi có bất kỳ đối tượng nào được tạo ra.  Constructor tĩnh cũng giống phương thức tĩnh nên không thể gọi các thuộc tính không phải static. |
| **Tính Chất Kế Thừa** |  | | **class <tên lớp con> : <tên lớp cha>**  {  }  Trong C#, **không hỗ trợ** đa kế thừa (1 lớp kế thừa từ nhiều lớp) những lại hỗ trợ thực thi nhiều interface (khái niệm về interface sẽ được trình bày trong bài [**INTERFACE TRONG C#**](https://www.howkteam.vn/redirect?Id=Y%2fHKZv6fbvNJnzXtx4Ic0n3nY4TfYJWT9kAQx46uZ%2f2tQYJMsmQ%2bEzebTh%2bkcLi4kzga29T%2fZF1oa%2fvFvOaP7LQjuN60OMxbHr1KjlYIWSDBTyhzjUFpuYMp1Pq%2bCttOGZ5Gl7%2fEBjbJWfG4DrrYUg%3d%3d)).  Các thành phần của lớp cha có được kế thừa xuống lớp con hay không là do phạm vi truy cập của thành phần đó là gì.   * Thành phần có phạm vi là private thì không được kế thừa. * Thành phần có phạm vi là protected, public thì được phép kế thừa.   Phương thức khởi tạo và phương thức huỷ bỏ không được kế thừa.  Nếu như lớp cha có phương thức khởi tạo có tham số thì đòi hỏi lớp con phải có phương thức khởi tạo tương ứng và thực hiện gọi phương thức khởi tạo của lớp cha thông qua từ khoá base.  **public <tên lớp>**(**<danh sách tham số của lớp con>**) **: base**(**<danh sách tham số>**)  {  } Vấn đề hàm trùng tên và cách gọi phương thức của lớp cha Trong C# có hỗ trợ từ khoá new nhằm đánh dấu đây là 1 hàm mới và hàm kế thừa từ lớp cha sẽ bị che đi khiến **bên ngoài**không thể gọi được.  **Cụ thể bạn sẽ thêm từ khoá new vào trước khai báo hàm Info() trong lớp Cat.**  **Nếu có 2 lớp info 1 của cha 2 của con thì mặc định sẽ gọi hàm info của con tr xog r mới tới cha** |
| **Tính Đa Hình**  **(lưu ý tính đa hình chỉ dc sử dụng khi có từ khóa Override** , nếu k có thì nó chỉ là kế thừa bt**)**  <https://www.howkteam.vn/course/lap-trinh-oop-voi-c/da-hinh-trong-lap-trinh-huong-doi-tuong-1380> | **Tính đa hình**là hiện tượng các đối tượng thuộc các lớp khác nhau có thể hiểu cùng 1 thông điệp theo các cách khác nhau.  Một ví dụ về đa hình trong thực tế. Ta có 3 con vật: chó, mèo, lợn. Cả 3 con vật này đều là động vật. Nhưng khi ta bảo cả 3 động vật kêu thì con chó sẽ kêu gâu gâu, con mèo sẽ kêu meo meo và con heo sẽ kêu ẹt ẹt.  Trong ví dụ trên 3 con vật: chó, mèo, lợn xem như là các đối tượng. Việc ta bảo 3 động vật kêu chính là thông điệp. Rõ ràng cả 3 con vật có thể hiểu cùng 1 thông điệp là kêu theo các cách khác nhau.  Từ khoá virtual và từ khoá override | | Để thể hiện được**tính đa hình**:   * Các lớp phải có quan hệ kế thừa với cùng 1 lớp cha nào đó. * Phương thức đa hình phải được ghi đè (override) ở các lớp con (sẽ được trình bày ngay sau đây).   **Virtual** là từ khoá dùng để khai báo 1 phương thức ảo (phương thức ảo là phương thức có thể ghi đè được). thường de ở lớp cha  **Override** là từ khoá dùng để đánh dấu phương thức ghi đè lên phương thức của lớp cha.  **Lưu ý:**   * Chỉ có thể ghi đè lên phương thức virtual hoặc abstract (sẽ trình bày ngay sau đây). * Tính đa hình chỉ được thể hiện khi đã ghi đè lên phương thức của lớp cha.   Vd cho dể hiểu: cha bạn cho bạn 1 ngôi nhà nếu như cha bạn có để từ khóa **Virtual**  thì bạn dc phép sử chữa theo ý bạn ngược thì không |
| **NÂNG CAO C#** | | | |
| **Collection** | Collection có thể hiểu là một mảng động vì nó có thể tang giảm số lượng phần tử 1 cách dễ dàng k giống như mảng bt nhưng collection chậm hơn mảng  LÚC NÀO NÊN DÙNG COLLECTION LÚC NÀO NÊN DÙNG MẢNG |  | |
|  |  |  | |
|  |  |  | |
|  |  |  | |
|  |  |  | |