PRN211

Basic Cross-Platform Application Programming With .NET

* Progress test 1: Quiz or HomeWork 10%
* Progress test 2: Assignment 10%
* 3 Workshop: 3 Assignment 10%
* 1 Group project: 25%
* PE: 25%
* FE: 30%

Project: thêm xóa sửa tìm kiếm, có kết nối CSDL (có 2 dạng mới cũ, mới điểm cao)

Ứng Dụng đa nền tảng chạy trên nhiều hệ điều hành khác nhau

Có thể tạo đc web thông qua ASP.NET MVP

Mô hình CLR

Viết code -> qua compiler phiên dịch lần 1 -> code bỏ vào file IL (Intermedia language) -> phiên dịch thêm lần nữa qua JIT (Just in time) Compiler -> thành native code (mã nhị phân) -> HĐH

Dùng ‘\_’ để ngăn cách giá trị dễ nhìn dành cho các con số

Khai báo parameter trong C#

Con trỏ -> dùng từ khóa ‘ref’

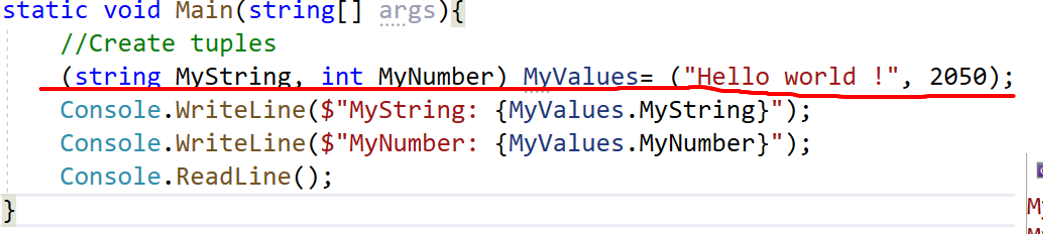
Từ khóa out-> trả giá trị ra chỗ đó

Params int[] list -> mảng vào nhập bao nhiêu phần tử cũng đc khi gọi hàm

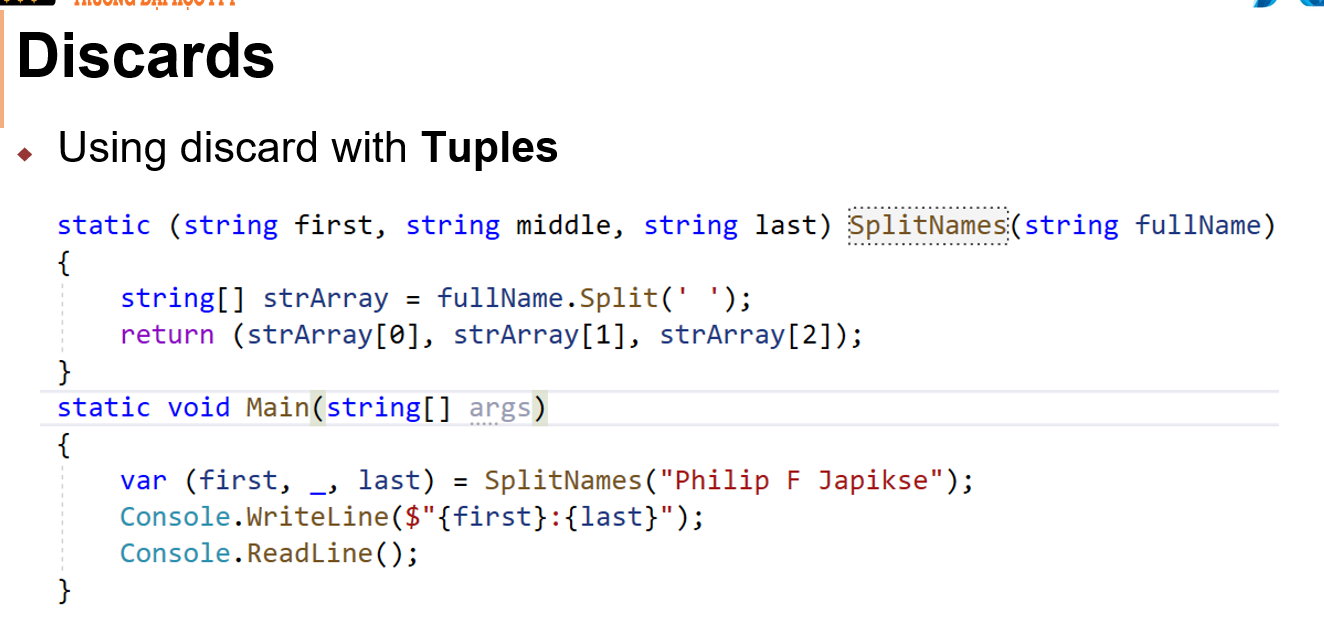
Ref có thể đưa vào return (biến trả về)

Hàm trong hàm (local function) kiểu dữ liệu trả về thông thường là giống nhau, đặt đâu cũng đc chỉ thằng cha dùng đc

Tuples: group các giá trị khác kiểu lại với nhau.



DISCARDS loại bỏ



IF và Switch case mở rộng

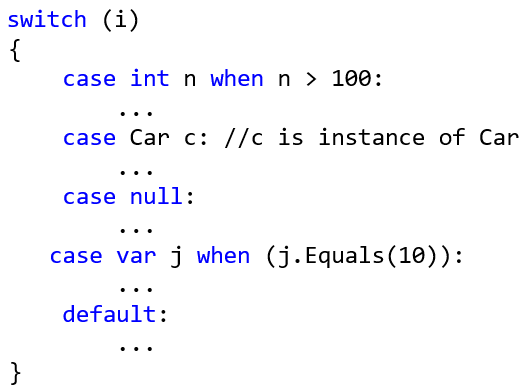
* **is** pattern
* The is pattern allows we to check whether an *input* variable is of a certain type, and then assign it to a new variable named *count*.

if (input is int count && count > 0)

* + This pattern can also be used to check if an *input* variable is null:

if (input is null

và đây là SWITCH:



**Null-Condition Operator**

* Khi muốn khai báo 1 biến có giá trị là null thì thêm dấu hỏi ngay sau kiểu dữ liệu
* Kiểm tra xem 1 cái có giá trị null dùng ??: trả về giá trị bên trái nếu ko null trả về giá trị bên phải nếu null

(numbers ??= new List<int>()).Add(5);

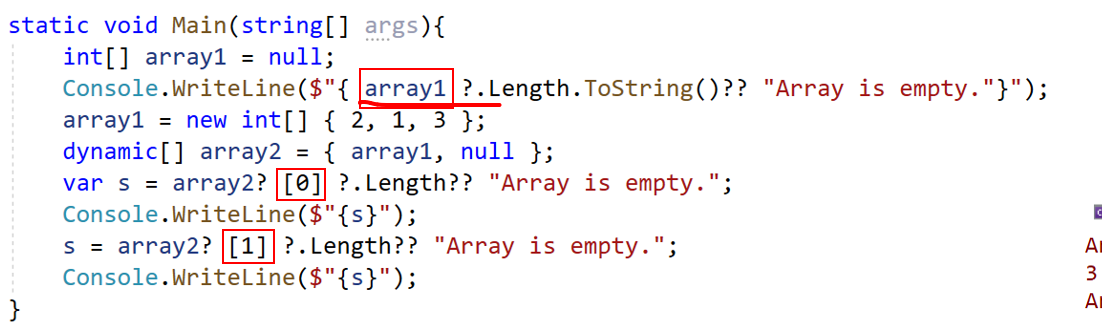
Console.WriteLine(string.Join(" ", numbers)); // output: 5

numbers.Add(a ??= 0);

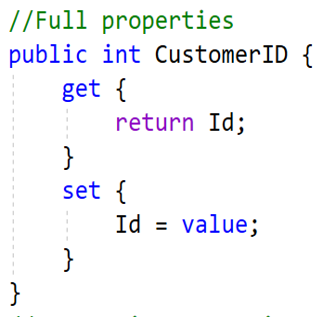
Console.WriteLine(string.Join(" ", numbers)); // output: 5 0

Console.WriteLine(a); // output: 0

* Riêng mảng thì tạo bình thường ko cần thêm ? nhưng khi test null thì phải thêm ? phía sau vd:



GETTER and SETTER của PROPERTIES của object được tạo cùng lúc (ĐÓNG GÓI)



thay đổi giá trị thì gán trực tiếp



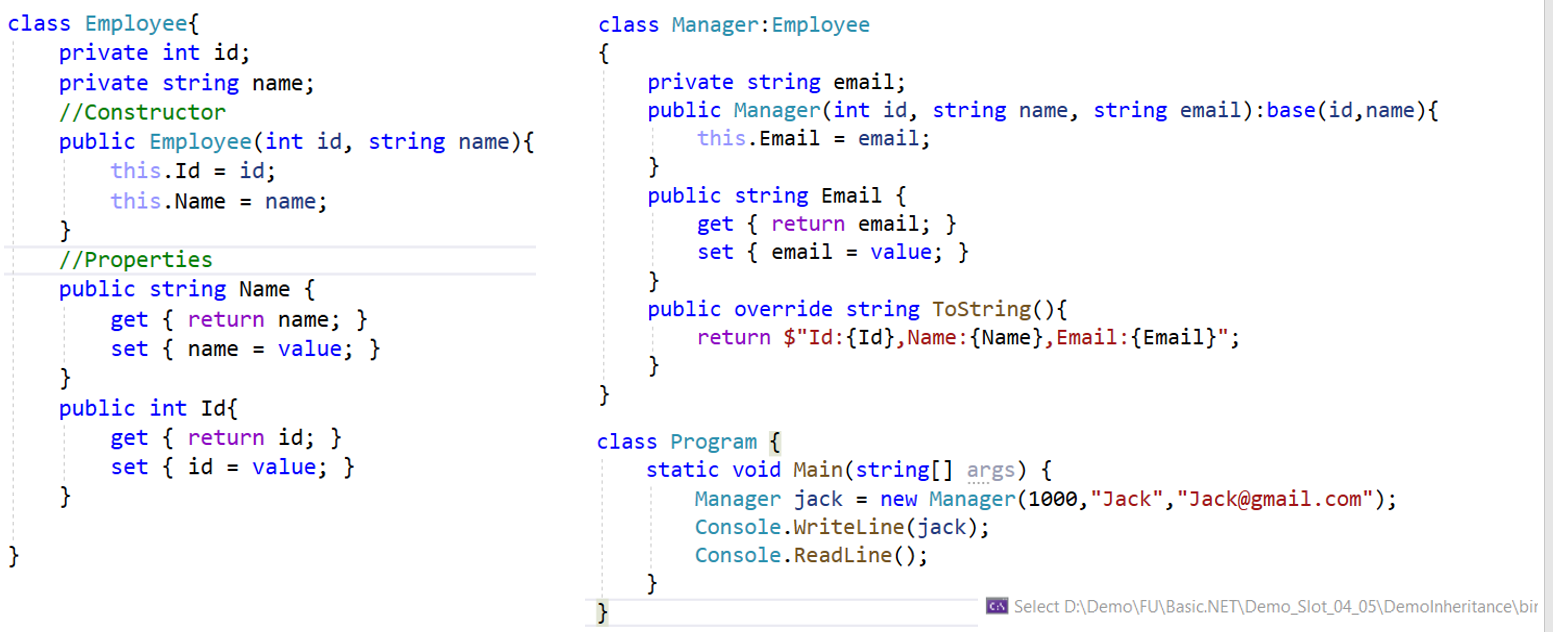
* Cú pháp tạo nhanh và gán default, nếu chỉ có get mà ko có set thì là read-only



* Nếu set mà đặt là init thì giá trị chỉ có thể gán khi tạo bằng cách đặt trong {}

Và không thể thay đổi sau này trừ khi object tự đổi

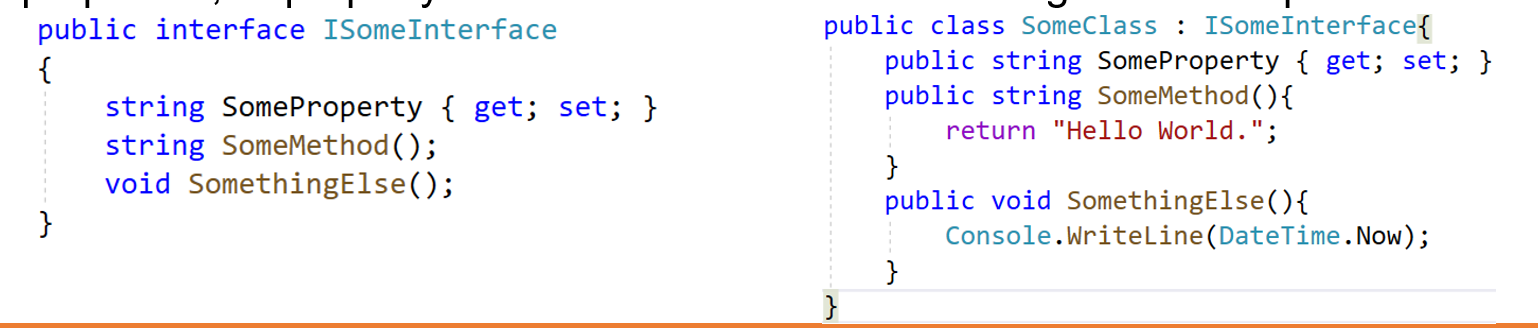
SYNTAX KẾ THỪA VÀ ĐA HÌNH





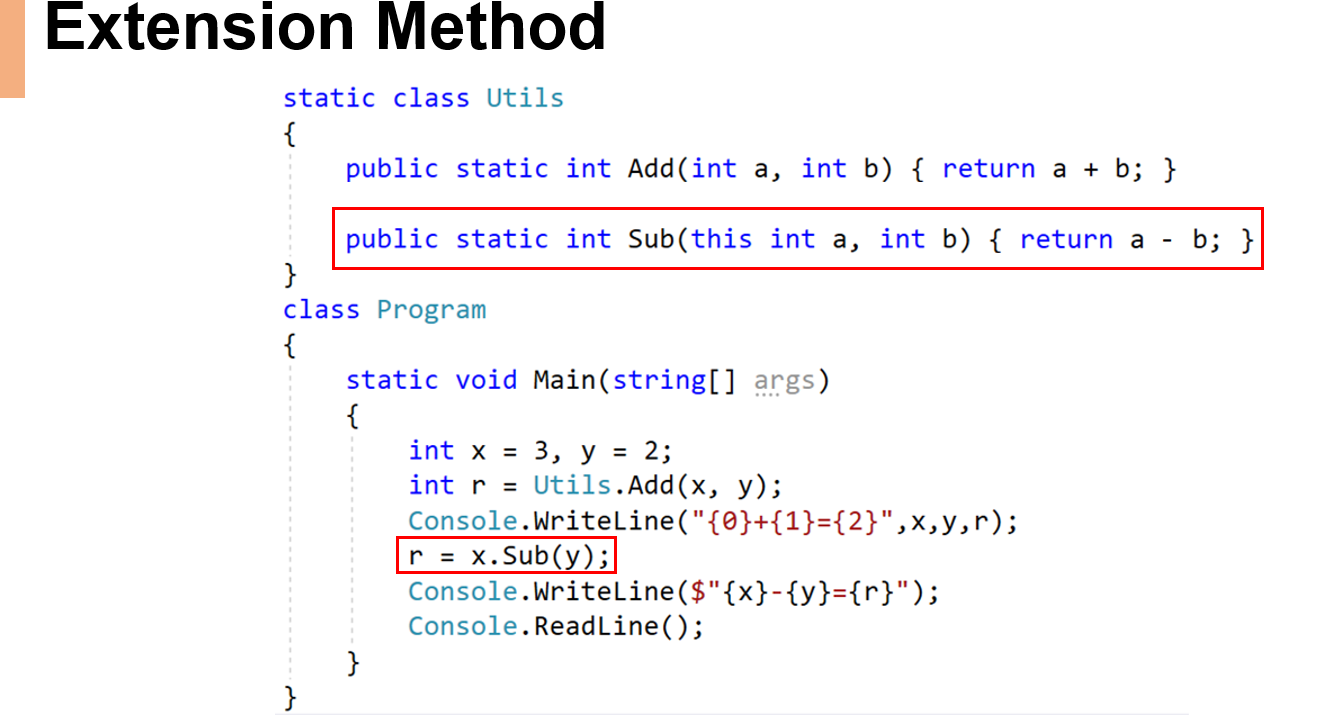
* Dùng dấu “:” biểu thị sự kế thừa -> Con:cha
* Contructor thì cũng vậy -> Con(toàn bộ Prop bao gồm cả cha đi kèm kiểu):Cha(properties contruct cha ko kèm kiểu)
  + Các prop của cha sẽ đc gán tương ứng theo thứ tự
* Method overide thì thêm overide vô

TÍNH TRỪU TƯỢNG:

* Virtual: có code ban đầu ko ép phải overide
* Abstract: không có code bắt con phải overide
* INTERFACE: -> kế thưa giống extends dùng dấu 2 chấm
* 
* THẰNG CON EXTENDS NHIỀU THÌ NGĂN CÁCH BẰNG DẤU PHẨY
* INTERFACE CÓ THỂ LÀ CON CỦA INTERFACE KHÁC

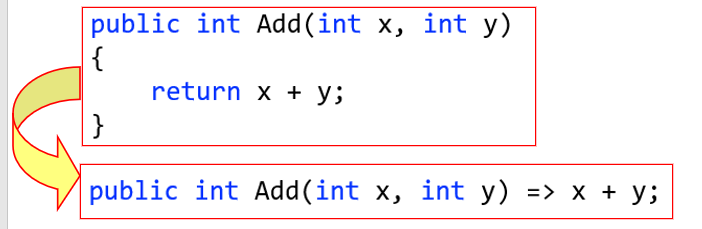
TOÁN TỬ is VÀ as

* Toán tử is dùng để kiểm tra kiểu dữ liệu của một object, trả về true nếu đúng false nếu sai
* Toán tử as dùng để kiểm tra kiểu dữ liệu của một object, trả về chính nó nếu đúng ko đúng thì trả về null

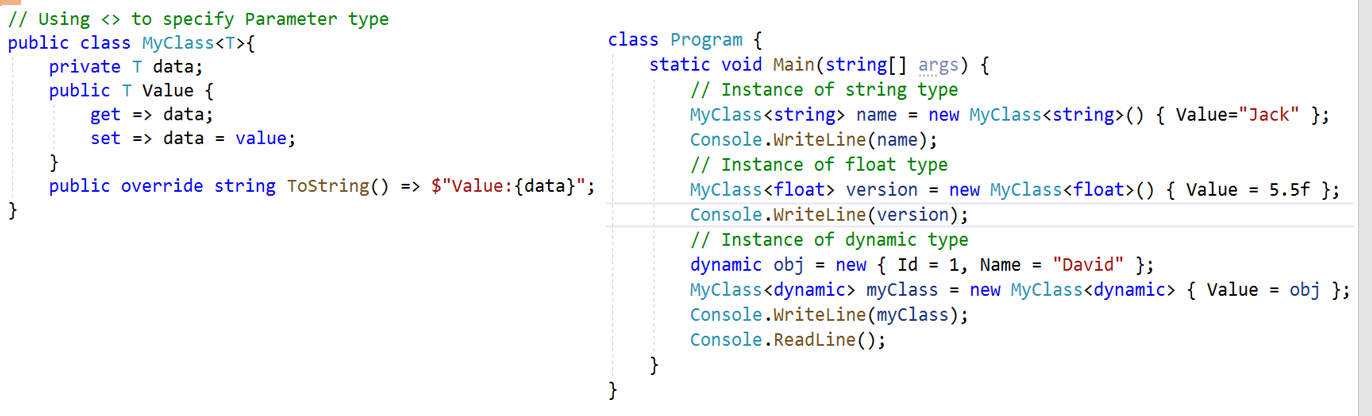
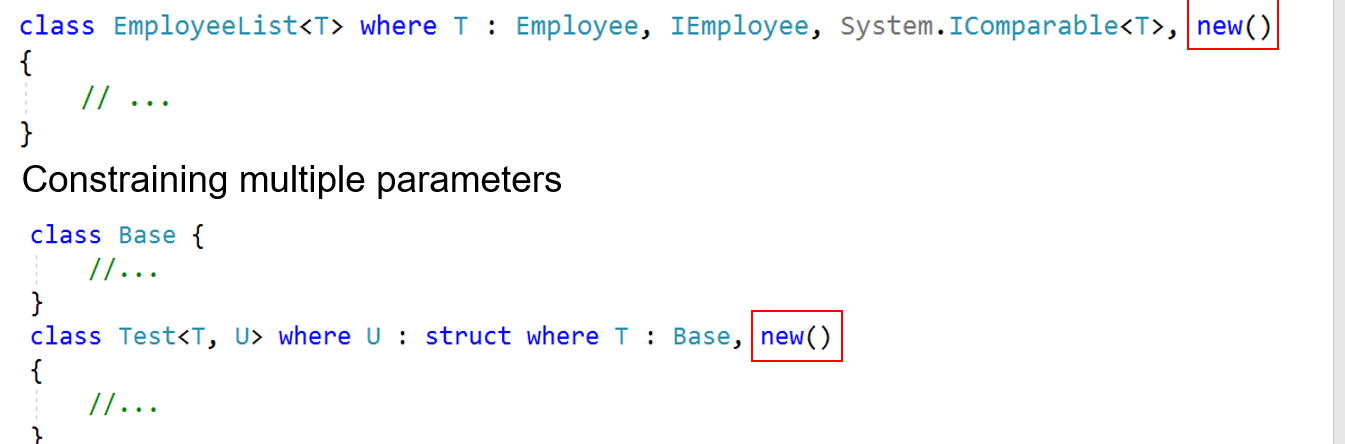


* This là thằng gọi đến cái method đó luôn, x.sub(y) -> sub(x , y);

HÀM VECTOR, dành cho các hàm siêu ngắn thường là return luôn



GENEGIC

* Một kiểu dữ liệu referent sẽ tốn vùng nhớ trên cả vùng nhớ heap và stack
* Quá trình boxxing và unboxing sẽ gây tốn dữ liệu và bộ dọn rác sẽ tự đi hủy rác trên vùng nhớ heap
* GENEGIC khai báo không care datatpye: Ký hiệu đặt trong ngoặc nhọn, datatype sẽ đc thế chỗ bằng chữ đó
* Khi dùng thì mới bắt đầu khai báo kiểu cho nó
* Ví dụ: 
* Datatype của biến này chỉ được khai báo khi truyền vào
* Tất nhiên để tránh truyền linh tinh thì ta phải ràng buộc dữ liệu như sau
* 
* Khởi tạo giá trị măc định
* 
* Tham chiếu thì null value thì là 0
* ->> ồ hóa ra các colllection đó h đầu dùng generic đó