Dotnet

* Dotnet là một nền tảng để xây dựng các ứng dụng khác nhau Vì vậy nó ko phải là 1 ngôn ngữ duy nhất mà tích hợp nhiều ngôn ngữ hợp tác với nhau để phát triển web, di động và ứng dụng
* Ứng dụng máy tính để bàn:
* Là những ứng dụng chạy trên máy tính để bàn các ứng dụng như máy tính, nodepad, accounting software, …Các ứng dụng này có thể có hoặc không tương tác với internet
* Các ứng dụng này cũng chạy độc lập và được cài trên máy người dùng

+ Bạn có thể thiết kế trang web thông qua sử dụng .net các ứng dụng web có thể xây dựng trên máy chủ và có thể truy cập từ máy khách thiết bị thông qua trình duyệt

VD: fb, youtube, … Vì chúng được cài đặt trên máy chủ và có thể truy cập từ máy khách thông qua URL

* Chúng sẽ tương tác với cơ sở dữ liệu

+ Bạn cũng có thể xây dựng mobile web thông qua .net

* Gồm 2 module là:
  + C#.net
  + ASP.net
* C#.net
* Tạo được Window GUI Application, Windows service, Console Application
* ASP.net
* Tạo được Website, Web application, Web service
* Ứng dụng đồ họa giao diện người dùng chạy trên local desktop gọi là Window GUI Application
* Console Application Là những ứng dụng chạy trên dòng lên ở command prompt
* Website hay còn gọi là trang web tĩnh cung cấp thông tin dựa trên dữ liệu tĩnh và khong được thay đổi thường xuyên
* Web application gọi là trang web động Nội dung được lấy trên database chứ ko phải tĩnh
* Web service là chương trình có thể tái sử dụng được cài đặt tại máy chủ và được truy cập từ máy khách
* Các chương trình C# hoặc VB.net đều sẽ được biên dịch theo 4 công đoạn
* Mã code trên ngôn ngữ gốc -> Mã ngôn ngữ trung gian được lưu trong file exe cách mở chạy visualstudio command promt gõ ildasm chọn file cần mở -> Để chuyển được file ngôn ngữ góc sang ngôn ngữ trung gian cần ILI là môi trường để chuyển đổi lúc này mã sẽ chuyển thành mã máy và trả ra output
* CLR là công cụ thực thi các chương trình mạng
* Là môi trường thực thi mã cho tất cả các ngôn ngữ mạng
* Là ngôn ngữ biết đổi ngôn ngữ IL sang ngôn ngữ máy
* CLR có các phần phụ của nó
  + Phần phụ thứ nhất: class Loader: Chịu trách nhiệm tải lớp và bộ nhớ trước khi thực hiện chương trình
  + Phần thứ 2: Memory manager: Chịu trách nhiệm phân bố bộ nhớ cho tất cả các đối tượng và biến cục bộ (Các biến cục bộ được lưu trữ trong stack, các đối tượng được lưu trữ trong heep)
  + Phần thứ 3: Garbage collector: Chịu trách nhiệm xóa các đối tượng được tạo bởi trình quản lý bộ nhớ
  + Phần thứ 4: Trình biên dịch JIT là đại diện cho trương trình biên dịch Justin Time chịu trách nhiệm chuyển đổi ngôn ngữ IL sang ngôn ngữ máy. Trình biên dịch sẽ không chuyển đổi hết cả chương trình mà chỉ chuyển đổi các mã được phần thứ nhất (class loader) tải được
  + Phần thứ 5: Exception manager: Trình quản lý ngoại lệ: Chịu trách nhiệm bắt tất cả các lỗi được nêu ra khi chương trình được thực hiện:

Bao gồm các thư viện như: SystemException, ApplicationException, StackOverflowException, InvalidOperatorException, SqlException, FormatException, …

* + Phần thứ 6: Thread manager(Trình quản lý luồng): Chịu trách nhiệm tạo, dừng các luồng.
  + Thread: là một đối tượng trong .NET chịu trách nhiệm thực thi mã nguồn
  + Phần thứ 7: Security manager(Quản lý bảo mật): Chịu trách nhiệm ngăn chặn chương trình truy cập các tài nguyên phần cứng
* .NET framework and its Architecture
* .NET framework là nền tảng được sử dụng để tạo ra một ứng dụng khác như window desktop hoặc ứng dụng web và browser
* Là một phần của .NET
* .NET được chia làm 2 phần: .NET framework and .NET Core
* .NET framework gồm 2 thành phần chính là: CLR (Common Language Runtime) and FCL(Framework Class Library)
* FCL: Chứa hầu hết các thư viện hoàn chỉnh có sẳn trong .NET
* Architecture:

Architecture

Common Language Runtime (CLR)

Base Class Library

ADO.NET

Others

WPF

ASP.NET

Win Form

Common Language Specification

Others

VC++.NET

VB.NET

C#

BCL: Base Class Library: Thư viện lớp cơ sở bao gồm tất cả các giao diện và các lớp được xác định từ trước và làm việc với các luồng (thread)

* FCL: Là một bộ nhiều BCL bao gồm tất cả các giao diện và các lớp mục đích chung ở tất cả các nền tảng: Console, Window, mobile, Application
* Vậy FCL = BCS + tất cả các thư viện trong Windows, Web application
* ADO.NET bao gồm tất cả các lớp được xác định trước có thể được sử dụng trong bất kì loại ứng dụng nào. Để kết nối với cơ sở dữ liệu và tương tác với cơ sở dữ liệu
* Winforms: là thư viện có sẳn trong Windows được sử dụng cụ thể để phát triển các thành phần GUI trong các ứng dụng window Forms.
* Để tạo các lớp, nhãn, Nút(Button) yêu cầu tạo các lớp có sẳn trong thư viện Winforms
* **Form, Label, Button, Textbox, … <= Winforms**
* WPF: là một cái mới được giới thiệu trong .NET 3.0 nâng cao hơn thư viện Winforms nhưng kĩ thuật và đồ họa cao hơn Winforms
* ASP.NET Là một thư viện hay một khung được sử dụng để phát triển các ứng dụng web Ở đây có các lớp xác định như:
* Page, Textbox, button, Checkbox, … <= ASP.NET=> Giao diện cho người dùng web
* ASP.NET có các thành phần phụ như: ASP.NET WebForms, ASP.NET Web pages, ASP.NET MVC, ASP.NET Web API, …
* Các thư viện others: WWF, WCF, …
* CLS: Không phải thư viện mà là tập hợp các quy tắc và khái niệm hoặc một số loại hướng dẫn chung cho tất cả các ngôn ngữ .NET
* Nói cách khác các ngôn ngữ C#, VB.NET, … đều được viết theo một số nguyên tắc nhất định. Những khái niệm phổ biến đó gọi là CLS
* Bao gồm: **CTS, Classes & Objects, References variable, Method paramete, Generics**
* CTS: Bao gồm tất cả các kiểu dữ liệu có sẳn trong ngôn ngữ .NET
* ASP.NET core cung cấp các thư frame work tạo lập trang web hỗ trợ trên nhiều nền tảng khác nhau
* Xamarin ứng dụng ASP.Net core để tạo ứng dụng di động cho android hoặc IOS, và thiết bị windows phone
* **OOP**
* Mục tiêu của lập trình hướng đối tượng là: Để nhóm dữ liệu và hoạt động của nó như một đơn vị có tên là đối tượng
* Đối tượng là một vật thể vật lý có thể đại diện cho con người và vật thật
* Class là một loại đối tượng không phải một tập hợp đối tượng
* Lớp mô tả danh sách các trường và phương thức mà bạn lưu trữ bên trong đối tượng
* Mọi đối tượng đều giữ một tập hợp giá trị các trường mà được quy định bởi lớp riêng của nó
* Phương thức là một tập hợp các câu lệnh để thực hiện một chức năng hay một hoạt động nào đó
* Trường tổng thể chỉ là về cách lưu giữ giá trị chứ không phải các thao tác
* Và các phương thức là để tính toán xử lý dữ liệu nhưng không phải để lưu trữ
* Tất cả đối tượng được lưu trữ trên “heap”
* Có nghĩa là khi chạy ứng dụng vùng nhớ sẽ được phân bố trên RAM
* Đối tượng là một lớp vật lý được tạo dựa trên lớp hiện có
* Cú pháp để khai báo 1 class
* Bên trái class luôn là từ khóa để chỉ loại sửa đổi truy cập (Access Modifier)
* Có 2 loại là public và internal
* Public: Duojcd truy cập trong class và các class khác
* Internal: được truy cập trong cùng assembly
* Bên cạnh phạm vi truy cập là loại class
* Có 4 loại:
* Static: Tĩnh chỉ gồm những thành viên tĩnh
* Là thành viên mà chỉ được khai báo 1 lần
* Abstract class: Lớp trừu tượng là lớp chứa các phương thức trừu tượng
* Sealed class: Lớp không bị kế thừa
* Multi partial class: là những lớp cùng tên nhưng được kết hợp bởi nhưng class đơn
* Class library: là một tập hợp các lớp sẽ được biên dịch dưới dạng DLL Thư viện liên kết động
* **Field**
* Trường là một biến được lưu trữ bên trong đối tượng
* Phải khai báo các trường trong lớp vì lớp là mô hình đối tượng
* Để khai báo một trường trong class cú pháp:
* AccessModifier modifier type fieldName
* **Có 6 loại accessModifier:** Internal, Public, Private, Protected, Private internal, Protected Internal
* Internal: Có thể truy cập trong class đó
* Public: Duojcd truy cập trong class và các class khác
* Private: Chỉ được truy cập trong cùng file và các thành viên và lớp sẽ ko được sử dụng trong class khác
* Protected: Là class mà chỉ được sử dụng trong class đó và các class kế thừa nó
* Private Internal: Chỉ có thể truy xuất đến thành viên và phương thức từ lớp kế thừa
* Protected Internal: Có thể truy xuất đến thành viên và phương thức từ lớp đó và lơp kế thừa
* **Có 3 loại modifier:** static, const, readonly
* Static: là chung cho tất cả đối tượng và có thể truy cập bằng tên class
* Const: Là giá trị của trường không thể thay đổi được, Và trình biên dịch sẽ tự động thay giá trị của trường đó khi biên dịch
* Readonly: Là trường mà giá trị không thể thay đổi đươc. Chỉ giới hạn thời gian biên dịch
* **Type: Là kiểu dữ liệu**
* Các giá trị của đối tượng sẽ được lưu trữ trong heap và các thể hiện của class sẽ chiếm bộ nhớ stack
* Lớp con là lớp mà có nguồn gốc từ một lớp khác đã có các trường
* Số file .cs trong một dự án là số hội đồng (amount of assembly)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Access modifier** | **In the same class** | **In the child class at the same assembly** | **In the other class at the same assembly** | **Child class at other assembly** | **Other class at other assembly** |
| **Private** | **V** | **X** | **X** | **X** | **X** |
| **Protected** | **V** | **V** | **X** | **V** | **X** |
| **Private Protected** | **V** | **V** | **X** | **X** | **X** |
| **Internal** | **V** | **V** | **V** | **X** | **X** |
| **Protected Internal** | **V** | **V** | **V** | **V** | **X** |
| **Public** | **V** | **V** | **V** | **V** | **V** |

* Một trường mà sử dụng phổ biến cho tất cả các đối tượng thì khai báo là static (trường tĩnh)
* Trường tĩnh được truy cập thông qua tên class
* className.fieldName
* Trường readonly là trường mà chỉ được cập nhật giá trị 1 lần
* **Methods**
* Phương thức là tập hợp các câu lệnh thực hiện một số quy trình hoặc hoạt động cụ thể dựa trên các trường trong lớp
* Phương thức là một thành viên của class. Nó không được lưu trữ trong đối tượng nhưng có thể được gọi từ đối tượng
* Một phương thức có thể chứa các tham số, các tham số ở đây là giá trị đến hoặc giá trị đầu cho phương thức
* Và ở cuối phương thức thường là kết quả trả về dựa trên kiểu dữ liệu trả về
* Giá trị trả về này sẽ được chuyển cho người gọi phương thức từ phương thức ban đầu được gọi
* Syntax:
* accessModifier modifier returnType methodName (parameter1, parameter2, …)
* accessModifier tương tự field
* modifier: static, virtual, abstract, override, new, partial, sealed
* **Đóng gói (Encapsolation) :** Là tất cả làm thế nào để nhóm các thành viên dữ liệu gồm 2 khía cạnh là module dữ liệu và ẩn dữ liệu
* **Module dữ liệu:** Nghĩa là dữ liệu của các trường sẽ được truy cập, cập nhật, sửa đổi dựa vào các phương thức chứ không được truy cập trực tiếp
* Sử dụng từ khóa get để truy xuất thông tin và sử dụng từ khóa set để cài đặt giá trị cho trường dữ liệu trong lớp
* Đóng gói ở đây có thể hiểu là đóng gói dữ liệu của trường trong các phương thức có thể sửa đổi bằng cách tạo các phương thức set hoặc get
* **Hiding data**: Là các thuộc tính field nên được khai báo private còn các phương thức nên được khai báo public
* Tất cả các biến được khởi tạo trong phương thức đều là biến cục bộ (local variable)
* Và các biến được truyền vào trong () của phương thức gọi là các tham số
* Các tham số là các biến có thể được nhận giá trị từ phần gọi của phương thức
* Các biến và tham số cục bộ sẽ được lưu trữ trên bộ nhớ stack
* Từ khóa **This** trong C# được gọi trước để đề cập đến đối tượng hiện tại
* Từ khóa này chỉ khả dụng cho những phương thức mà non-static Nhưng không cho các phương thức tĩnh. Sử dụng từ khóa this để phân biệt các phương thức mà có tham số truyền vào và trường nhận giá trị là giống nhau
* Phương thức tĩnh
* Một lớp có thể chứa 2 thể loại phương thức là phương thức tĩnh và phương thức thể hiện (instance methods)
* Bất cứ khi nào muốn thao tác với các trường thể hiện, bạn nên đi theo phương thức thể hiện
* Nhưng muốn thao tác với trường tĩnh thì nên đi theo phương thức tĩnh
* Trường tĩnh và phương thức tĩnh không được đưa vào các đối tượng, chúng độc lập với các đối tượng và chung cho tất cả đối tượng

|  |  |
| --- | --- |
| **Phương thức thể hiện** | **Phương thức tĩnh** |
| * Liên kết với đối tượng * Khai báo mà không cần từ khóa static * Chỉ được gọi thông qua đối tượng * Mục đính là thao túng các trường thể hiện | * Liên kết với lớp * Khai báo với từ khóa static * Được gọi thông qua tên class * Mục đích là thao túng các trường tĩnh |

* Nếu khai báo đối tượng mà ko có new className() thì biến tham chiếu đến đối tượng luôn giữ giá trị NULL
* Phương thức tĩnh luôn truy cập được trường tĩnh và ko truy cập được trường thể hiện
* Còn Phương thức thể hiện thì truy cập được cả 2
* Đối số mặc định là giá trị t gán vào tham số bên trong phương thức (default arguments)
* Syntax: methodName (parameter = defaultValue)
* Giá trị mặc định này được sử dụng khi t gọi phương thức mà không truyền đối số. Còn nếu đối số được truyền thì giá trị mặc định được bỏ qua
* Cách đặt tên cho tham số trong phương thức để gắn giá trị mặt định
* Syntax: methodName( parameterName:value, parameter2Name: value)
* Có nghĩa là một phương thức có nhiều tham số mặc định
* VD: **methodName(parameter1 = value, …)**
* Khi gọi chúng ta có thể tùy chọn truyền vào tham số mà chúng ta muốn
* Những nếu một dự án lớn thì 1 phương thức gồm nhiều tham số thì khi truyền giá trị như vậy sẽ gây khó đọc cho những người khác nên việc đặt tên cho giá trị truyền vào đối số là điều cần thiết. Điều này được thực hiện khi phương thức được gọi
* Syntax: **objectName.methodName(parameter2: value)** Có nghĩa là chúng ta truyền vào giá trị đối số cho tham số thứ 2 còn các tham số còn lại sẽ lấy giá trị mặc định (default Arguments)
* Overloading methods
* Có nghĩa là có nhiều phương thức giống tên nhưng tham số trong phương thức đó là khác nhau
* Bộ sửa đổi tham số
* **Ref, out, in, params**
* Bộ điều chỉnh tham số là công cụ sửa đổi có thể áp dụng cho tham số của phương thức
* Trình sửa đổi tham số xác định cách nhận giá trị tham số từ phần gọi
* Điều này có nghĩa là nó xác định xem giá trị đối số có được nhận giá trị từ phần tử gọi phương thức hay ko
* Khi không truyền vào bộ điều chỉnh tham số cho tham số trong phương thức thì bộ điều chỉnh là mặc định
* Có nghĩa là mọi thay đổi của tham số trong phương thức đó sẽ ảnh hưởng đến biến đối số
* Vì biến số và đối số được khai báo trong stack
* Ref
* Là công cụ sửa đôi tham số thứ 2 trong C#
* Có nghĩa là khi bạn thực hiện những thay đổi đối với tham số thì đối số truyền vào cũng được thay đổi theo
* Syntax: accessModifier modifier type methodName(ref typeVar varName, …)
* Khi gọi hàm methodName (ref argu)
* Điều kiện là biến argu phải được đặt trước
* Out
* Có nghĩa là giá trị đối số sẽ không được truyền cho biến tham số nhưng ngược lại giá trị tham số sẽ được gán trở lại vào đối số
* Syntax: tương tự ref
* accessModifier modifier type methodName(out typeVar varName, …)
* Calling methodName(out argu)
* Điều kiện tiên quyết khi sử dụng out là phải khởi tạo giá trị cho biến tham số trong phần thân phương thức nếu không sẽ bị lỗi biên dịch
* Có thể khai báo biến đối số trong lúc gọi hàm khi thực hiện công cụ sửa đổi ‘out’
* Syntax: Calling methodName(out type argu) và sau đó biến argu sẽ được tạo và giá trị của tham số cuối phương thức sẽ được truyền lại cho argu như trên và biến argu có thể sử dụng ở ngoài phương thức.
* In
* Cộng cụ sửa đổi in được sủ dụng như sau: Giá trị đối số sẽ được truyền cho biến tham số nhưng giá trị biến tham số sẽ trở thành readonly (có nghĩa là giá trị biến tham số sẽ ko được sửa đổi trong phần thân phương thức)
* Syntax:
* accessModifier modifier type methodName(in typeVar varName, …)
* Calling methodName(in argu)
* Ref return
* Ref return là khả năng của một phương thức trả về một biến tham chiếu
* Syntax: ref type methodName(){return ref var;}
* Calling method: ref var = ref methodName();
* Lợi ích của việc sử dụng ref return là khi cập nhật giá trị cho var thì giá trị của trường trong class cũng được cập nhật theo
* Params
* Được sử dụng để nhận một tập giá trị dưới dạng mảng
* Syntax:
* accessModifier modifier type methodName(varName, …)
* Calling methodName(params type[] arrayName)
* Và có thể truy cập các phần tử đối số bằng chỉ mục của mảng
* Local Function
* Chức năng cục bộ được sử dụng lồng trong phương thức
* Có nghĩa là định nghĩa 1 function như 1 hàm trong C++ nhưng dược khai báo và sử dụng trong cùng 1 phương thức
* Phân biệt phương thức và hàm:
* Phương thức là cái lớn đại diện cho 1 chức năng lớn của đối tượng
* Còn hàm đại diện cho 1 phần nhỏ của một chức năng lớn
* Và hàm cục bộ chỉ có thể được gọi trong phương thức
* Và hàm cục bộ trong phương thức có thể sử dụng các tham số của phương thức
* Static local function
* Có nghĩa là hàm trong phương thức được khai báo tĩnh ‘static’
* Điều đặc biệt của hàm tĩnh là không thể truy cập các tham số của phương thức
* Recursion (đệ quy)
* Gọi lại tên hàm trong hàm đang cài đặt khi thỏa mãn điều kiện
* Type Conversion (Chuyển đổi kiểu dữ liệu)
* Có 4 kỹ thuật chuyển đổi:
  + Implicit casting
  + Explicit casting
  + Parsing/TryParse
  + Conversion method
* Implicit casting
* Là loại chuyển đổi từ kiểu số sang kiểu số cao hơn
* VD: int => long
* Việc chuyển đổi được thực hiện ngầm
* Tức là khi ta gán giá trị của biến kiểu dữ liệu của 1 biến nhỏ sang kiểu dữ liệu của một biến lớn
* VD: sbyte(1 byte) a = 10; double(8 byte) b = a (điều này hợp lệ)
* Explicit casting
* Là loại chuyển đổi từ loại số cao hơn sang kiểu số thấp hơn
* VD Long => int
* Double => short
* Explicit casting là kiểu chuyển đổi mà thêm kiểu dữ liệu cho biến được gán cho biến bị gán
* Syntax: a = (type)b;
* Điều này có thể dẫn đến sự mất mát dữ liệu nếu vùng giá trị của biến được ép kiểu quá lớn
* Parsing/ TryParse
* Là loại chuyển đổi từ kiểu dữ liệu chuỗi sang kiểu dữ liệu số
* Parsing => Syntax: DestinationDataType.Parse(SourceValue);
* TryParse => Được sử dụng để kiểm tra chuỗi ban đầu có được chuyển đổi thành công hay không.
  + Trường hợp giá trị nguồn hợp lệ thì kết quả trả về là true
  + Trường hợp giá trị nguồn không hợp lệ trả về false
* Syntax: DestinationType.TryParse(SourceValue, out variableName)
* Cách 2: DestinationType.TryParse(SourceValue, out destinationType variableName)
* Conversion method
* Là kiểu chuyển đổi từ bất kì kiểu dữ liệu nguyên thủy nào sang kiểu dữ liệu nguyên thủy nào
* VD: string => other type
* Other type => string
* Syntax: type DestinationVariable = Convert.ConversionMethod(SourceValue)
* VD: Convert.ToInt32(sourceValue)
* Chú ý: ToInt64 = long
* ToSingle = float
* ToBoolean = bool
* ToInt16 = short
* ToUInt16 = ushort
* Kĩ thuật này có thể làm xuất hiện 1 ngoại lệ
* Thuật ngữ Abruptly Terminated (Chấm dứt đột ngột)
* Constructor
* Phương thức đặc biệt của class. Chứa logic khởi tạo của các trường. Cũng chứa các khởi tạo bổ sung
* Khi từ khóa new được gọi là lúc mà constructor được thực hiện
* Có 2 loại: Constructor Rõ ràng, Constructor ẩn
* Syntax: accessModifier modifier className(parameter){}
* Có thể chứa 1 hoặc nhiều tham số tùy chọn
* accessModifier gồm 6 thuộc tính (private, protected, …)
* modifier: static
* Không được áp dụng các trình sửa đổi khác cho constructor
* accessModifier nên đặt là public hoặc là internal
* Không có phương thức trả về
* Có thể tạo nhiều hàm constructor trong cùng 1 lớp
* Có 2 kiểu constructor
  + Tạo constructor cho trường hiển thị (instance field)
  + Tạo constructor cho trường tĩnh (static field)
* Instance constructor
* Được tạo tự động khi 1 đối tượng được khởi tạo
* Default accessModifier: Private
* Có 2 loại:
  + Implicit Constructor
  + Explicit Constructor
* Implicit Constructor: Được gọi là hàm tạo không tham số. Cung cấp sự khởi tạo các trường với các giá trị mặc định
* Explicit Constructor: Được gọi là hàm tạo có tham số, cung cấp sự khởi tạo các trường bằng cách gán giá trị cho các trường đó
* Static constructor
* Được khởi tạo 1 lần khi đối tượng đầu tiên được khởi tạo
* Default accessModifier: Public
* Overloading Constructor
* Là việc có nhiều hàm khởi tạo trong 1 class với số lượng tham số của mỗi constructor là khác nhau
* Khi bạn sử dụng default constructor thì có thể thêm các giá trị cho các trường mà default constructor đó chưa khởi tạo
* VD default constructor: Khởi tạo giá trị cho MSSV
* Mà khi khởi tạo bạn muốn thêm trường tenSV và BirthDay vào
* Syntax: SV sv1 = new SV(){tenSV = value, BirthDay = value2};
* Properties and Indexers
* Thuộc tính là các tính chất của trường trong class. Ko giống như phương thức vì không có giá trị tham chiếu(). Trong thuộc tính là các hàm như get, set để gán dữ liệu và lấy dữ liệu
* Syntax: AccessModifier modifier type propertyName

{

Set{field = value;}

Get{return field}

}

* Thuộc tính giúp nâng cao tính đóng gói của class. Không thể truy cập trường từ bên ngoài
* Thuộc tính và phương thức chỉ là một phần của class. Không chiếm bất kỳ bộ nhớ nào.
* Constructor chỉ có thể lấy giá trị của thuộc tính gọi là constructor readonly
* Constructor chỉ có thể gán giá trị cho thuộc tính gọi là constructor writeOnly
* Indexer là tất cả những thứ được sử dụng để truy cập phần tử bên trong mảng của 1 trường trong class
* Syntax:
* AccessModifier modifier type methodName this[int index]
* {

Set{

This.nameProperties = value;

}

Get{

Return nameProperties;

}

* }
* Indexer overload Có nghĩa là sử dụng nhiều hàm indexer để lấy giá trị của trường thông tin tùy vào parameter truyền vào
* **Set accessor:**

+ Được sử dụng để xác thực giá trị đến và gán giá trị tương tự vào trường

+ Thực thi tự động khi một giá trị được gán vào thuộc tính

+ Có tham số mặc định (ẩn) được gọi là giá trị, đại diện cho giá trị hiện tại tức là được gán cho thuộc tính

+ Không thể có bất kì tham số bổ sung nào

+ Nhưng cũng không thể trả lại bất kì giá trị nào

* **Get accessor:**

+ Được sử dụng để tính toán giá trị và trả về cùng giá trị của trường hiện tại

+ Thực thi tự động khi thuộc tính được lấy

+ Không có tham số ngầm định

+ Nên trả về giá trị của trường

* **Properties:**

+ Thuộc tính tạo lớp bảo vệ xung quanh các trường ngăn việc gán các giá trị không hợp lệ vào thuộc tính và cũng thực hiện một số hành động khi ai đó gọi thuộc tính

+ Không có bộ nhớ được phân bố cho thuộc tính

* Thuộc tính tự động triển khai là thuộc tính mà không có khai triển cho get và set được sử dụng để tạo thuộc tính mà chỉ để gán và lấy giá trị ko cần kiểm tra
* **Indexer**

+ Không được truyền ref or out vào trong bộ chỉ mục

+ Trình lập chỉ mục không được khai báo static

+ Trình chỉ mục có thể được overloaded

* **INHERITANCE**
* Lớp cha không được truy cập vào trường, phương thức hay là thuộc tính của lớp con
* Có nghĩa là lớp con có thể có hết các thành viên của lớp cha và có thể bổ sung thêm thành viên của lớp con
* Có 5 loại kế thừa trong C#
  + Single inheritance (Kế thừa duy nhất)
  + Multi-level inheritance (các lớp kế thừa nhiều tầng)
  + Hierarchical inheritance (1 chame, nhiều con)
  + Multiple inheritance (1 lớp kế thừa nhiều lớp)
  + Hybrid inheritance (Là kế thừa kết hợp 2 hay nhiều loại kế thừa khác)
* Sử dụng đa kế thừa bằng cách khai báo interface vào class
* Từ khóa Base được sử dụng để truy cập thuộc tính, trường và phương thức của lớp cha
* Hàm tạo: hàm tạo của lớp con được sử dụng để khởi tạo các trường của lớp con và hàm tạo của lớp cha được sử dụng để khởi tạo các trường của lớp cha
* Vì vậy khi khởi tạo 1 đối tượng của lớp con cần gọi hàm tạo của lớp cha
* Phương pháp ẩn là quá trình ẩn một phương thức của lớp cha bằng một phương thức khác trong lớp con
* Cú pháp là đặt tên phương thức trong lớp con giống lớp cha giống cả tham số lẫn kiểu trả về và thêm từ khóa new trước loại trả về
* Phương thức overriding (Ghi đè) là quá trình mở rộng phương thức của lớp cha bằng cách tạo một phương thức cùng mọi thứ với lớp cha
* Ý nghĩa của việc mở rộng là khi bạn gọi phương thức thông qua đối tượng của lớp con cả phương thức của lớp con và phương thức của lớp cha sẽ được thực thi
* Cú pháp sử dụng từ khóa virtual ở phương thức của lớp cha và sử dụng từ khóa override trong phương thức của lớp con
* Truy cập phương thức của lớp cha bằng cách sử dụng từ khóa Base
* Syntax: base.methodName()
* Khi sử dụng method overriding đầu tiên phương thức của lớp cha sẽ được thực thi rồi đến phương thức của lớp con được thực thi
* Trên thực tế: Trình tự thực thi đến với lớp con đầu tiên nhưng là một câu lệnh đầu tiên cuả phương thức của lớp con. Mà câu lệnh đầu tiên của phuonwg thức của lớp con là Base.methodName nên phương thức của lớp cha sẽ được thực thi
* Không có từ khóa virtual thì không có phương thức override
* LỚP NIÊM PHONG Là một lớp không thể được kế thừa
* Syntax: sealed class NameClass
* Có thể kế thừa từ lớp khác. Có nghĩa là có thể là lớp con của một lớp nào đó
* Có thể kế thừa từ giao diện khác
* Được sử dụng cho mục đích bảo mật
* PHƯƠNG THỨC GHI ĐÈ NIÊM PHONG : Giả sử bạn có phương thức ảo trong lớp cha và bạn có 1 phương thức ghi đè trong lớp con. Bây giờ bạn không muốn phương thức bị ghi đè bởi các lớp con khác. Lúc đó sử dụng phương thức ghi đè
* Syntax: Class cha:
* accessModifier virtual type methodName()
* Class con
* accessModifier sealed override type methodName()
  + - * **Abstract and interfaces**
* Sự trừu tượng là một khái niệm về lập trình khách quan
* Được xem như 1 class cha cho tất cả các class có cùng bản chất.
* Abstract class sẽ định nghĩa tất cả các phương thức để các lớp dẫn xuất kế thừa từ nó phải định nghĩa lại.
* Tất cả các phương thức lớp abstract phải được khai báo public hoặc protected
* Có thể có thuộc tính nhưng không được sử dụng từ khóa abstract cho thuộc tính
* Một abstract class không cho phép bạn khởi tạo object từ lớp đó
* Trừu tượng có thể làm bằng 2 cách
  + Lớp trừu tượng (Abstract class)
  + Interfaces
* Mục đích chính của lớp trừu tượng là cung cấp một tập hợp các trường, thuộc tính và phương thwucs chung cho tất cả các lớp con tương ứng của nó. Nhưng lớp trừu tượng trực tiếp không thể được khởi tạo để gọi các phương thức tương ứng.
* Vì vậy muốn sử dụng các trường hay phương thức của lớp trừu tượng thì phải khởi tạo đối tượng lớp con rồi sau đó mới có thể truy cập và sử dụng các trường hay phương thức của lớp trừu tượng
* Phương thức trừu tượng là phương thức được sử dụng để khai báo chứ không được định nghĩa. Được sử dụng khi một lớp chỉ muốn định nghĩa phương thức nhưng muốn khai triển trong lớp con. Lúc đó lớp trừu tượng được sử dụng
* Syntax:
* accessModifier abstract returnDataType MethodName(parameter);
* Vì vậy phương thức trừu tượng là phương thức được khai báo trong lớp cha nhưng không chứa thân phương thức
* Lớp con khi viết phần thân của phương thức trừu tượng phải sử dụng từ khóa override
* Syntax:
* accessModifier override returnDataType MethodName(parameter)
* Ở đây từ khóa override không có nghĩa là mở rộng phương thức của lớp cha mà là tuân thủ nguyên tắc của phương thức trừu tượng
* Nếu có một tham số, tên hoặc là một thành phần gì đó khác lớp trừu tượng được định nghĩa trong lớp cha thì phương thức đó không phải là phương thức hợp lệ
* Hạn chế là tên phương thức số lượng tham số và kiểu trả về của phương thức lớp con và lớp cha phải giống nhau
* Chỉ có thể khai báo phương thức trừu tượng trong lớp trừu tượng
* Nếu có phương thức trừu tượng được định nghĩa trong lớp trừu tượng cha thì bắt buộc lớp con phải viết phần thân của phương thức trừu tượng đó nếu không viết sẽ gây ra lỗi trình biên dịch
  + - * **INTERFACE**
* Giao diện là một lớp trừu tượng thực tế. Chỉ chứa các phương thức trừu tượng
* Interface là tập hợp các lớp trừu tượng mà phải được thực hiện bởi các lớp con
* Syntax: interface interfaceName { methodName…}
* Các phương thức được khai báo trong interface mặc định chế độ điều chỉnh là public
* Khi các lớp con kế thừa interface thì các lớp con phải khai triển phần thân của phương thức. Và khai báo public
  + - * Các quy tắc của giao diện
* Lớp con triển khai giao diện phải triển khai tất cả các phương thức của giao diện. Các phương thức mặc định là công khai và trừu tượng
* Không thể tạo đối tượng cho giao diện
* Có thể tạo biến tham chiếu cho giao diện
* Có thể lưu địa chỉ của đối tượng của bất kì lớp con nào khác trong cùng một giao diện
* Một lớp con có thể có nhiều giao diện cha. Bất kể khi nào một lớp con có nhiều giao diện cha thì đều được gọi là đa kế thừa
* Một giao diện có thể là một giao diện con của một giao diện mẹ khác được gọi là kế thừa giao diện
* Tất cả phương thức trong giao diện đều là phương thức trừu tượng mặc dù có khai báo từ khóa abstract hay không
* Các trường hợp nên sử dụng interface: Khi bạn chỉ muốn cung cấp các phunowg thức mà không muốn thực hiện khai triển của phương thức và trong lớp chỉ chứa các phương thức trừu tượng không chứa các phương thức bình thường thì khi đó nên sử dụng giao diện
  + - * TÍNH ĐA HÌNH (POLIMOPHISM)
* Tính đa hình là khả năng của một câu lệnh gọi các phương thức khác nhau trong các tình huống khác nhau
* Điều đó có nghĩa là dựa trên tham chiếu đối tượng hoặc dựa trên các đối số mà bạn cung cấp. Nó có thể gọi các phương thức khác nhau
* Đa hình còn được hiểu là khả năng ra quyết định
* Nếu quyết định được đưa ra bởi trình biên dịch tại thời điểm biên dịch thì đa hình đó gọi là đa hình thời gian biên dịch
* VD: Method overloading
* Nếu quyết định đó được CLR thực hiện tại thời điểm thực thi tùy thuộc vào trạng thái bộ nhớ trong thời gian chạy, Thì nó được gọi là đa hình thời gian chạy
* VD: Method overriding
* Đa hình thời gian biên dịch còn được gọi là đa hình tĩnh hay liên kết sớm
* Vì quyết định được thực hiện sớm hơn. Tức là trước khi chạy
* Còn đa hình thời gian chạy còn được gọi là liên kết muộn vì được thực hiện trong lúc thực thi
* Tính đa hình động có thể được thực hiện bằng 2 cách: Hoặc với các lớp trừu tượng hoặc với giao diện trừu tượng
* Lớp cha có thể giữ tham chiếu đến đối tượng của lớp con
* Ví dụ:
* ParentClassName c1;
* C1 = new childClass1();
* C1.MethodChildClass1
  + - * Interface Inheritance
* Khi 2 interface kế thừa nhau nếu trong interaface con có phương pháp là n và số phương pháp của interface mẹ là m thì số lượng phương pháp mặc định là m + n
* Trong trường hợp lớp kế thừa nhiều interface nhưng trong 2 interface có cùng phương thức thì nếu lớp con chỉ triển khai tên phương thức thì phương thức đó áp dụng cho cả 2 interface
* Còn nếu muốn triển khai cho mỗi interface một thân phương thức khác nhau thì cần thêm tên interface.nameMethod(parameter) trước.
* Khi có 2 phương thức như vậy thì khi khai báo vd khai báo interface1 tham chiếu đến class thì khi đối tượng đó gọi phương thức thì sẽ triển khai phương thức của interface1 và tương tự cho trường hợp khác
  + - * Explicit interface implementation
* Được sử dụng để triển khai một phương thức giao diện riêng tư. Có nghĩa là phương thức của giao diện sẽ trở thành riêng tư đối với class con
* Nếu lớp con kế thừa 2 giao diện có cùng phương thức, kiểu trả về và tham số thì sử dụng explicit interface implementation để triển khai 2 phương thức khác nhau cho 2 giao diện
  + - * NAMESPACE
      * Create Namespace
* Không gian tên là tập hợp các lớp và các kiểu khác nhau như cấu trúc, giao diện, kiểu đại diện, kiệu liệt kê, v.v
* Namespace is a collection of classes and “other types such as interfaces, tructures, delegate type, enumerations)
* Thực tế thường nhóm tất cả các lớp và các loại khác thành một đơn vị duy nhất nhằm mục đích cụ thể.
* Có thể truy cập class hoặc interface thuộc namespace bằng cú pháp:
* namespaceName.className
* Khi làm dự án thực tế cần tách class riêng interface riêng thành cái tệp
  + - * Không gian tên lồng nhau
* Khi có một tập hợp lớn các lớp hoặc interface trong cùng một không gian tên. Khi đó chúng ta muốn gom nhóm các class con trong namespace thì namespace lồng nhau được sử dụng
* Mục tiêu của không gian tên lồng nhau là tạo một nhóm con bên trong một không gian tên lớn
  + - * Sử dụng từ khóa using để khai báo sử dụng namespace
* Được sử dụng để truy cập class và các kiểu khác của namespace trong cùng một tệp mà không cần phải khai báo tên vùng tệp
* Thị Using được sử dụng để nhập không gian tên giúp dễ dàng truy cập vào các lớp và interface là một phần của không gian tên
* Nếu như số lần sử dụng namespace đó quá nhiều thì mới nên khai báo còn không thì có thể truy cập thông qua namespaceName.className
* Sử dụng bí danh: Là cách để gán một tên namespace dài cho một tên bí danh để code trở nên ngắn gọn dễ hiểu hơn
* Syntax:
* Using NameAlias = namespaceName
* Và khi sử dụng như vậy thì các đối tượng của class con của namespace phải được khai báo lại
* Tên bí danh còn dược sử dụng trong trường hợp mơ hồ về tên
* Tức là 2 namespace đều có cùng class name thì để phân biệt đang sử dụng class nào thì ta nên sử dụng tên bí danh để truy cập
  + - * Sử dụng tĩnh
* Bất cứ khi nào bạn khai báo static namespaceName thì sử dụng đó gọi là sử dụng tĩnh
* Mục tiêu của sử dụng tĩnh là bạn có thể truy cập đến tất cả thuộc tính và phương thức tĩnh của class tĩnh mà không cần đề cập đến tên class
* Lợi ích tổng thể là mã được đơn giản hóa
  + - * Partial classes
* Là lớp được trải dài trên các tệp
* Có nghĩa là lớp được chia làm nhiều phần, mỗi phần được xác định trong mỗi tệp
* Mục tiêu là chia class thành nhiều phần cho nhiều người phát triển ví dụ:
* Người 1 phát triển phần 1, người 2 phát triển phần 2
* Chức năng cho phép chia nhỏ và cho mỗi người phát triển một phần
* Bất kì thuộc tính trừu tượng hay niềm phòng (sealed) được áp dụng trong một lớp từng phần sẽ tự động áp dụng trong tất cả các lớp từng phần còn lại có cùng tên
* Lợi ích nữa là giúp mã được tự động trong tệp khác là lợi ích tốt nhất của lớp từng phần
  + - * Phương thức từng phần
* Là phương thức được định nghĩa trong file1 và được định nghĩa phần thân trong file2. Mặc định thì phương thức từng phần là private nên không thể truy cập từ bên ngoài.
* Phương thức từng phần không thể trả về giá trị vì vậy chỉ có từng phần kiểu void
* Vì vậy để sử dụng phương thức từng phần ở bên ngoài thì ta cần phải tạo thêm một phương thức gọi lại phương thức từng phần
  + - * Lớp tĩnh
* Mục tiêu của lớp tĩnh là tránh tạo thành viên thể hiện trong một lớp
* Và cũng để tránh tạo đối tượng cho một lớp cụ thể
* Trong lớp tĩnh chỉ có trường tĩnh, phương thức tĩnh và thuộc tính tĩnh
* Lợi ích là không thể tạo một thể hiện của lớp tĩnh
* Bởi vì các thành phần đều là tĩnh nên có thể truy cập mà không cần đến thể hiện của lớp
* Bất cứ khi nào muốn tránh việc vô tình tạo ra các thành viên thể hiện trong một lớp và cũng tránh việc vô tình tạo các đối tượng của một lớp cụ thể thì khi đó bạn nên khai báo lớp tĩnh
  + - * Enumeration
* Bất cứ khi nào bạn muốn một trường hoặc một biến có thể chấp nhận bất kỳ một trong các hằng số nào thì bạn sẽ đi đến khai báo enum
* Bạn không muốn các trường haowjc biến có thể chứa giá trị khác ngoài những giá trị này thì bạn nên sử dụng enum
* Giúp người dùng nhận dạng đúng dữ liệu được nhận vào và giúp tránh nhập sai dữ liệu
* Syntax:
* Public enum enumName
* {
  + Value1,
  + Value2,
  + Value3
* }
* Cách sử dụng enumName.value1
* Và trong enum thì giá trị cũng được liệt kê theo thứ tự 0 1 2
* Bạn cũng có thể thay đổi giá trị index trong enum bằng cách gán
* Syntax:
* Public enum enumName:int|short|long nhưng không được sử dụng string
* {
  + Value1 = 100,
  + Value2,
  + Value3
* }
* Khi bạn khai báo như vậy thì giá trị index của enum sẽ bắt đầu chạy từ 100 và tăng lên đối với các giá trị sau.
  + - * Kiểu giá trị (Value types)
* Structure and enumeration là các kiểu giá trị
* Dùng để lưu các giá trị đơn giản nhất
* Thể hiện không phải là đối tượng
* Giá trị được lưu trong stack
  + - * Kiểu tham chiếu (Reference type)
* String, class, interface, delegate là các kiểu tham chiếu
* Dùng để lưu trữ các giá trị phức tạp
* Thể hiện của nó là một đối tượng
* Giá trị được lưu trong bộ nhớ heap
  + - * Cấu truc (structure)
* Các thể hiện của cấu trúc là thể hiện cấu trúc hoặc biến cấu trúc không phải là đối tượng cấu trúc vì vậy thể hiện này sẽ được lưu trữ trong bộ nhớ stack
* Không thể tạo phương thức khởi tạo ít tham số cũng như phương thức hủy cho structure bởi vì bản thân ngôn ngữ đã cung cấp một phương thức khởi tạo ít tham số cho mọi cấu trúc gán giá trị mặc định
* Đó là tất cả các kiểu số thì gán 0, string null, boolean false
* Nhưng có hỗ trợ phương thức khởi tạo hàm tạo có nhiều tham số
* Không hỗ trợ kế thừa
* Cấu trúc có thể triển khai 1 hoặc nhiều giao diện
* Không thể khởi tạo phương thức ảo hay trừu tượng vì không có kế thừa
* Cấu trúc nhanh hơn các lớp vì các cấu trúc được lưu trữ trong ngăn xếp
* Không thể gán giá trị ban đầu cho trường trong structure
* Không hỗ trợ quyền sửa đổi protected hoặc internal protected
* Kiểu cấu trúc không thể gán giá trị null
* Nếu bạn có hàm tạo nhiều tham số trong struct thì bạn phải khởi tạo tất cả các trường cho structure nếu không sẽ bị lỗi
* Khi gán giá trị cấu trúc 1 cho cấu trúc 2 thì khi thay đổi giá trị của cấu trúc 1 thì cấu trúc 2 không bị thay đổi
* Khi gán giá trị đối tượng 1 cho đối tượng 2 thì khi thay đổi giá trị của đối tượng 1 thì giá trị của đối tượng 2 cũng bị thay đổi
  + - * Readonly Structure
* Cấu trúc chỉ đọc là cấu trúc mà tất cả các thành phần của cấu trúc phải ở dạng chỉ đọc bao gồm trường, thuộc tính và phương thức
* Bạn có thể khởi tạo các giá trị của các trường trong hàm tạo
* Có nghĩa là phương thức của cấu trúc chỉ đọc chỉ chứa hàm get
* Lợi ích của cấu trúc chỉ đọc là cải thiện hiệu suất điều này có nghĩa là chúng hoạt động nhanh hơn các cấu trúc thông thường vì mọi thứ đều ở trạng thái chỉ đọc
  + - * Cấu trúc của các kiểu nguyên thủy
* Primitive types: Ngoại trừ String còn tất cả các ngôn ngữ khác đều là hiện thân của cấu trúc
  + - * System.Object
* Là một lớp được xác định trước và nó là lớp cơ sở của tất cả kiểu giá trị và kiểu tham chiếu
* Lớp kế thừa lớp system.Object gồm System.ValueType, classes, Interface
* Lớp kế thừa lớp System.ValueType là Structure và Enumeration
  + - * Kế thừa System.Object
* Gồm các phương thức như:
  + Virtual bool Equals(object value) So sánh 2 đối tượng
  + Virtual int GetHashCode() Trả về địa chỉ ô nhớ của đối tượng htai
  + Type GetType()
  + Virtual String ToString()
* Có thể kế thừa và ghi đè
* Phương thức Type không phải là phương thức ảo nên không thể ghi đè và nó trả về tên kiểu có nghĩa là tên lớp mà đối tượng được tạo bao gồm cấu trúc, lớp và giao diện
* Còn ToString thì là phương thức ảo thông thường nó trả về tên lớp Nhưng bạn có thể ghi đè và cho nó trả thông tin tùy thích
  + - * Boxing
* Là quá trình chuyển đổi từ một kiểu giá trị sang kiểu tham chiếu
* Và unboxing là ngược lại từ loại tham chiếu sang loại giá trị
* Boxing là quá trình sao chép giá trị của một biến kiểu nguyên thủy vào một đối tượng con của System.Object giá trị được chuyển từ stack và heap và địa chỉ được lưu trong stack
* Có 2 cách để khởi tạo một đối tượng System.Object
  + System.Object nameObject
  + Object nameObject
    - * Unboxing
* Là quá trình chuyển đổi từ kiểu tham chiếu sang kiểu giá trị
* Nó chỉ xảy ra khi các dữ liệu tương thích
* Có nghĩa là nếu kiểu nguyên thủy là con của kiểu tham chiếu thì có thể unboxing
* Việc unboxing nội bộ sẽ không diễn ra mà bạn phải truyền cú pháp ngoại lệ
* **Syntax:**
  + **Datatypesource variable = (datatypesource)valuesource**
    - * Generics;
* Lớp chung chung là lớp mà kiểu dữ liệu của các trường có thể thay đổi
* Có nghĩa là khi khởi tạo một đối tượng cho lớp chung cần truyền kiểu dữ liệu làm đối số
* Syntax:
* **Class className<T>**
* **{**
  + **Public T fieldName;**
* **}**
* Lợi ích là chúng ta có thể lưu trữ nhiều loại trường
* Syntax khởi tạo:
* ClassName<datatype> referenceName = new className<datatype>()
* Được sử dụng khi bạn không biết chính xác kiểu dữ liệu của đối tượng và trường. Cho phép đối tượng quyết định kiểu dữ liệu
* Có thể sử dụng nhiều kiểu dữ liệu chung cho lớp chung bằng cú pháp:
* **Class className<T1, T2>**
* **{}**
* Những ràng buộc chung trên lớp chung là những kì vọng và hạn chế với các tham số chung. Áp dụng ràng buộc trên một tham số chung cụ thể
* Cú pháp để viết các ràng buộc là:
* **Class className<T> where T : class(struct, interfaceName, className, new())**
* Vì vậy có thể gây ra lỗi thời gian chạy nếu bạn cung cấp không đúng kiểu dữ liệu
* Nếu ràng buộc không xuất hiện mà bạn ép kiểu thì có thể gây ra lỗi compiler
* Vì vậy việc sử dụng ràng buộc là bắt buộc nếu muốn chấp nhận một class cụ thể hoặc class con của nó nhưng không phải điều khác
* Nếu lớp đó có 2 tham số kiểu chung thì để gán ràng buộc cho 2 tham số đó chúng ta có thể sử dụng cú pháp
* **Class className<T1, T2> where T1:… where T2: …**
* **{}**
* Phương thức chung:
* Là phương thức có thể nhận kiểu dữ liệu làm tham số chung, Các phương thức chung có thể có 1 hoặc nhiều tham số chung
* Vì vậy bất cứ lúc nào không chắc chắn kiểu dữ liệu truyền vào cho phương thức chúng ta có thể sử dụng phương thức chung
* Khởi tạo phương thức: **public void methodName<T>**
* Gọi phương thức: **MethodName<datatype>(valueHere)**
* Không cần phải sử dụng class chung mà cũng có thể sử dụng phương thức chung
  + - * Tại sao cần lưu trữ giá trị null
* Được sử dụng khi bạn muốn gán một giá trị không giá trị cho một đối tượng nào đó thì có thể nói rằng giá trị đó là null
* Chỉ có những đối tượng mới có kiểu null
* Còn struct và enumeration thì mặc định là không có giá trị null
* Có thể chuyển đổi kiểu non-nullable sang nullable bằng cách thêm dấu ? vào sau kiểu dữ liệu
* Cách 2: Nullable<datatype> variableName = null
* Có thể kiểm tra một kiểu giá trị có đang lưu giữ giá trị hay không bằng cú pháp **nameObject.Namefield.HasValue**
* Khi đã chuyển đổi một kiểu giá trị sang một kiểu tham chiếu (nullable) thì chỉ có thể truy cập giá trị của biến đó bằng cú pháp .Value
* Có 1 biến x đang được chuyển từ int sang int?
* Khi ta muốn gán giá trị của biến x cho biến y thì cú pháp phải làm là:
* **Int y = x.Value;**
  + - * NullCoalescing
* Là toán tử được sử dụng để chuyển đổi giá trị thực hiện như một phép so sánh
* Cú pháp:
* valueName??ValueIfNull
* Có nghĩa là nếu biến là null thì giá trị bên phải sẽ được trả về và nếu ko null thì chính giá trị của biến đó sẽ được trả về
* Cú pháp trên được sử dụng để tránh viết câu lệnh if dài dòng
* Nhưng có một hạn chế của cú pháp này là giá trị thay thế phải cùng loại với giá trị đang so sánh null
  + - * Null propagation Operator (toán tử truyền dẫn null) (?. Và ?[]
* Chỉ có thể truy cập vào trường, thuộc tính, phương thức của biến tham chiếu cụ thể khi tham chiếu khả biến không bằng null
* Được sử dụng để kiểm tra xem một đối tượng có null hay không
* Lợi ích là tránh nullReferenceExecution khi sử dụng trường hay thuộc tính của một đối tượng
* **objectName?.fieldName**
* **=**
* **(objectName.fieldName ==null)? Null:objectName.fieldName**
  + - * Extension Method and Pattern Matching
* Phương thức mở rộng là phương thức được viết bên ngoài lớp nhưng được đưa vào lớp một cách hợp lí
* Phương thức mở rộng được sử dụng khi bạn không có mã nguồn của một lớp để thay đổi
* Mặc dù được báo ở ngoài class nhưng nó vẫn là một thành viên của thư viện class
* Các trường hợp dùng phuonwg thức mở rộng là thêm phương thức vào lớp khai báo hoặc thư viện lớp
* Để triển khai một phương thức tĩnh cần khai báo một lớp tĩnh
* Bên trong phải có một phương thức tĩnh và phải có tham số là từ khóa this className làm tham số
* Khi bạn khởi tạm phương thức mở rộng dưới dạng phương thức tĩnh nhưng khi gọi lại phương thức đó trong program thì có thể dùng instance để gọi phương thức đó và không cần truyền bất kì tham số nào
* Khi bạn khai triển phương thức mở rộng có thể tăng tính teamwork giữa các thành vien trong nhóm
* Phương thức mở rộng không hỗ trợ override nhưng có thể mở rọng với phương thưc snghieem phong sealed
* Lớp tĩnh mà khởi tạo không được là lớp trong parameter
  + - * Pattern matching (Khớp mẫu)
* Được sử dụng khi một lớp con kế thừa một lớp chà và khi khởi tạo đối tượng lớp cha là new lớp con thì không thể truy cập vào phương thức hay thuộc tính của lớp con được khi đó chúng ta sẽ sử dụng pattern matching
* Hữu ích khi có một đối tượng thể hiện lớp cha nhưng không biết đối tượng được tạo cho lớp con hay lớp cha
* Và điều này còn giảm thiểu đoạn mã code
* CÚ pháp
* **If(parentObjectName is childclassName classChange)**
* **{**
  + **Body truy cập thuộc tính lớp con theo biến classChange**
* **}**
* **=**
* **If(parentObjectName is childClassName)**
* **{**
  + **Body nhưng trước khi truy cập phải khởi tạo biến lớp con và chuyển đổi kiểu từ parentClass sang childClass**
  + **childClass objectName = (childClass)parentObjectName;**
* **}**
* Cú pháp này hữu ích cho việc chuyển đổi kiểu tham chiếu từ lớp cha sang lớp con
  + - * Implicitly typed variable
* Sử dụng từ khóa var để khai báo dữ liệu
* Var không phải là kiểu dữ liệu mà là kiểu để cài đặt dữ liệu
* Khi sử dụng var để khai báo biến thì trình biến dịch sẽ tự động kiểm tra kiểu dữ liệu của biến được gán vào biến khởi tạo và ép kiểu cho biến khởi tạo
* Lợi ích của việc sử dụng từ khóa var: Giúp nhà phát triển thay vì viết cụ thể một kiểu dữ liệu dài dòng, phức tạp thì có thể sử dụng từ khóa var để rút ngắn code
* Còn được sử dụng khi nhà phát triển không biến được chính xác kiểu dữ liệu sẽ trả về thì khi đó từ khóa var được sử dụng sẽ hữu ích
* Điều luật của implicitly typed là khi khởi tạo biến phải khởi tạo giá trị hoặc đối tượng cho biến đó
* Var không được sử dụng làm kiểu dữ liệu của tham số, kiểu trả về hay kiểu dữ liệu của trường trong class, interface, structure
* Không thể gán giá trị null cho biến được khai báo var
  + - * **Dynamically Typed Variable**
* Có thể sử dụng từ khóa dynamic để khởi tạo biến. Khi sử dụng từ khóa động thì có thể gán giá trị hoặc không
* Có thể gán bất kì kiểu dữ liệu nào vào biến động
* Các biến được khai báo kiểu động có thể thay đổi kiểu dữ liệu trong suốt chương trình
* Được sử dụng khi không chắc kiểu dữ liệu mà mính muốn lưu trữ
* Điều quan trọng là trình biên dịch sẽ không kiểm tra kiểu dữ liệu đối với các biến động
* Vì vậy khi gọi các phương thức hay thuộc tính của biến đó trình biên dịch sẽ không kiểm tra được thuộc tính hay phương thức đó có tồn tại hay không
* Hạn chế sử dụng biến động vì nó không cung cấp intellisense Có nghĩa là không cung cấp gợi ý method hay thuộc tính phải xác định thủ công
* Có thể sử dụng cho kiểu tham số, trường, hay kiểu trả về
  + - * **Inner class**
* Lớp bên trong là lớp được định nghĩa bên trong một lớp khác
* Cú pháp:
* Class nameclass
* {
  + Class innerclassName
  + {
  + }
* }
* Vì vậy các thành viên của lớp bên trong chỉ có thể truy cập trong cùng một lớp trong
* Nếu lớp trong được khai báo public thì thành viên của lớp bên trong có thể được truy cập bởi lớp bên ngoài
* Mục đích của lớp bên trong là nhóm các tập hợp các lớp có liên quan đến mục đích cụ thể
* Lớp ngoài không thể truy cập thành viên của lớp trong nếu không có đối tượng của lớp trong
* Nó có thể giúp gói gọn các chức năng cụ thể thành một class giúp dễ đọc và dễ bảo trì
  + - * Garbage collection
* Trong CLR Trình quản lý bộ nhớ là nơi chịu trách nhiệm tạo các đối tượng và trình thu gom rác có trách nhiệm thu gom rác
* Kích thước heap mặc định là 64MB phụ thuộc vào phiên bản CLR
* Quá trình thu gom rác liên quan đến việc xác định tất cả các đối tượng được lưu trữ trong đống và cũng xác định tất cả các tham chiếu của đối tượng tương ứng
* Trình tự động thu gom rác sẽ tự động được gọi khi không còn dung lượng trống để tạo ra đối tượng mới
* Có thể gọi phương thức GC.Collect()
* Khi gọi phương thức trên CLR sẽ tự động xác định các đối tượng không được tham chiếu và xóa chúng trong bộ nhớ
* C# cung cấp 2 trình thu gom rác là destructor và dispose
* Sự khác nhau giữa 2 trình thu gom này là destructor được gọi tự động khi kết thúc chương trình còn dispose sẽ được thực thi theo yêu cầu
* Idisposable là một giao diện được xác định trước trong không gian tên hệ thống và tạo ra phương thức được xác định trước gọi là Dispose
* Có thể tạo đối tượng với Idisposable bằng cú pháp
* Using (ClassName referenceVariable = new className())
* {}
* Sau khi kết thúc câu lệnh Using thì hàm dispose sẽ được gọi
  + - * Delegate
* Đại biểu là một đối tượng đặc biệt lưu trữ một tham chiếu của một hoặc nhiều phương thức thỏa mãn các tham số và kiểu trả về được định nghĩa bởi kiểu ủy nhiệm
* Sử dụng ủy quyền có thể gọi một hay nhiều phương thức một cách gián tiếp
* Đại biểu là tất cả về truy cập gián tiếp của một phương thức
* Đại biểu là khối xây dựng để tạo nên sự kiện
* Nói cách khác sự kiện là các đại biểu nội bộ
* Các kiểu ủy nhiệm có nguồn gốc nội bộ từ một lớp cha được xác định trước gọi là system.Delegate
* Cú pháp khởi tạo đại diện và đối tượng đại diện:
* Khởi tạo đại diện:
* Public delegate ReturnType DelegateTypeName(Param1, …)
* Khởi tạo đối tượng
* DelegateTypeName ReferenceVariable = MethodName;
* Sau đó có thể gọi phương thức bằng cách sử dụng phương thức gọi của kiểu đại biểu
* referenceVariable.Invoke(arg1, …)
* Kiểu đại diện có thể gọi bất kì phương thức nào miễn là cùng số lượng tham số và kiểu trả về
* Multiple Delegate là đối tượng lưu trữ nhiều phương thức có cùng kiểu trả về và số lượng tham số
* Có thể sử dụng toán tử += để thêm ủy quyền khi sử dụng ủy quyền nhiều phương thức
* Không nên sử dụng kiểu giá trị trả về cho ủy nhiệm nhiều phương thức vì khi sử dụng ủy nhiệm nhiều phương thức thì giá trị trả về của phương thức 1 có thể ko được trả lại như ý muốn thay vào đó nên sử dụng kiểu void
* Sự kiện chỉ có nhiệm vụ chuyển thông báo từ lớp này sang lớp khác
* Sự kiện bao gồm 2 lớp là lớp sản xuất và lớp người đăng kí
* Lớp sản xuất là lớp khởi tạo và nâng cao sự kiện
* Lớp người đăng kí sẽ là lớp triển khai các phương thức và thực hiện phương thức khi sự kiện được nâng cao
  + - * **Array**
* Cách khai báo:
* Type[] arrayReferenceVariableName = new type[size]
* Theo mặc định các mảng được lưu trữ trong heap
* Mảng theo mặc định được coi là đối tượng
* Biến lưu trữ mảng là lưu địa chỉ của mảng
* Sự khác biệt giữa for và foreach là for dựa vào vị trí index còn foreach dựa vào trình tự của phần tử
* Foreach chậm hơn for
* IndexOf Xuất ra chỉ mục của giá trị trong mảng. Nếu giá trị ko tồn tại lưu -1
* Cú pháp để tìm kiếm vị trí của giá trị kể từ vị trí nhất định trong mảng theo cú pháp: Array.IndexOf(ArrayName, FindValue, startindex)
* Tìm kiếm nhị phân cũng được sử dụng trong array. Để sử dụng được tìm kiếm nhị phân thì mảng phải được sắp xếp tăng dần
* Để gán giá trị cho mảng 2 chiều thì phải gán giá trị cho 1 hàng {1, 2, 3}
  + - * Mảng răng cưa
* Khác với mảng 2 chiều bình thường là mảng răng cưa thì số cột ở mỗi hàng có thể khác nhau
* Cú pháp khai báo:
* Type[][] arrayReferenceVariable = new type[rowSize][]
* arrayReferenceVariable[indexRow] = new type[size]
* Để xuất ra mảng răng cưa thì phần tử của mảng có tham chiếu là a[i][j]
  + - * Collection
* Là một tập hợp có chứa nhóm giá trị cùng loại hoặc khác loại. Có thể lưu trữ các phần tử đồng nhất và các phần tử không đồng nhất. Có thể lưu trữ các giá trị nguyên thủy hoặc không nguyên thủy.
* Có nghĩa là có thể tạo ra tập hợp giá trị. Hoặc có thể tạo nhóm đối tượng của lớp cụ thể
* Có thể thêm hoặc bớt các phần tử trong list như vector trong C++
* Bộ sưu tập chung chỉ có thể lưu trữ các phần tử cùng kiểu dữ liệu
* Bộ sưu tập không chung chung có thể lưu trữ nhiều hơn 2 kiểu dữ liệu
* Lợi ích:
  + Số lượng phần tử không giới hạn
  + Có thể thêm xóa sửa bất cứ lúc nào
  + Có thể xác định chính xác kích thước của danh sách
  + Có thể tìm kiếm sao chép, …
  + Được sử dụng không không biết kiểu lưu trữ là kiểu dữ liệu gì và không biết kích thước chính xác của mảng thì nên dùng collection
    - * List
* Được sử dụng nhiều nhất trong C#. Sử dụng list collection bất cứ khi nào muốn lưu trữ cùng kiểu dữ liệu
* Được phát triển nội bộ bằng cách sử dụng lớp danh sách
* List được khai báo trong system.collection
* Có thể truy cập kích thước của list bằng list.count
* Cơ chế hoạt động của list là mỗi khi thêm hoặc xóa 1 phần tử thì nó sẽ tự động xóa mảng cũ và tạo 1 mảng mới rồi thêm phần tử vào trong nội bộ
  + - * Add, AddRange
* Cú pháp Add -> myList.Add(value) Value cùng kiểu giá trị với kiểu giá trị của list
* Phương thức addRange là phương thức được sử dụng để thêm 1 collection vào 1 collection khác
* Cú pháp AddRange -> myList.AddRange(otherList)
* Những điểm cần lưu ý khi sử dụng AddRange là kiểu dữ liệu phải cùng với kiểu dữ liệu ban đầu
  + - * Insert và InsertRange
* Cú pháp insert: -> myList.insert(index, value)
* Phần tử ở vị trí Index sẽ bị đẩy xún và thêm value vào vị trí đó
* Cú pháp insertRange -> myList.insertRange(index, otherList)
* Tương tự add.
  + - * Remove, RemoveAt, RemoveAll, RemoveRange, Clear
      * **Ienumerator**
* Mục đích là đại diện cho nhóm các phần tử

A screen shot of a computer

Description automatically generated

**Interface** Inumerator có các phương thwucs: Current trả về đối tượng hiện tại, MoveNext trả về true false nếu có đối tượng tiếp theo

* **Iterator**
* Public Ienumerable <Type> yield return ;
* Ko là constructor hoặc destructor
* Ko thể chứa ref hoặc out
* Có thể có nhiều yield return
  + - * Anonymous type
* Là loại đặc biệt cho phép tạo đối tượng mà ko cần tên kiểu cụ thể. Được tự động định nghĩa tại thời điểm runtime. Được sử dụng khi muốn tạo 1 đối tượng nhanh mà ko cần định nghĩa class hay struct
* Đặc điểm:
  + Readonly
  + Auto generate (Có sẳn get và set)
    - * Nested Anonymous type
* Là đối tượng anonymous được lồng vào nhau. Có nghĩa là có thể tạo 1 anonymous type trong 1 anonymous type khác
* Ưu điểm
  + Ko cần tạo class
  + Tiện lợi khi dùng LINQ
* Hạn chế
  + Ko tái sử dụng
  + Khó bảo trì
    - * Anonymous Array
* Là mảng ko cần khai báo trước kiểu dữ liệu một cách tường minh và thường chứa các phần tử kiểu vô danh. Thuận tiện trong các trường hợp ngắn hạn và ko cần sử dụng lại
* Ưu điểm
  + Tiết kiệm thời gian
  + Dễ hiểu
* Hạn chế
  + Ko thể tái sử dụng
  + Chỉ đọc
  + Khó bảo trì