

Course Introduction

Course Introduction

Learning Path

The diagram illustrates the learning path for C#. It starts with a central blue box labeled 'C#' which branches into two categories: 'Basic' and 'Intermediate'. The 'Basic' category leads to a list of 7 topics: 1. Introduction (history, version, hello world...), 2. Exe / DLL - Runnable C# file, 3. Basic OOP, 4. Declaration and Assignment, 5. Operators, 6. Array, and 7. Flow Controls. The 'Intermediate' category leads to a list of 6 topics: 1. Advance OOP (static, sealed class, delegate,...), 2. Date and String, 3. Collection and Generics, 4. Exception, 5. IO, and 6. Concurrency.

C# Fresher Academy

Mục tiêu khóa học:

Course Introduction

Objectives

The diagram illustrates the objectives of the course. A central blue box labeled 'Objectives' branches into three main goals: 1. Deeply understand C# core concepts, 2. Querying the data using LINQ, and 3. Implementing MultiThreading and Asynchronous programming. The 'Deeply understand C# core concepts' goal is further expanded to include: OOP (Dynamically Typed Objects, Optional and Named Parameters, Out variables, Tuples, Pattern Matching).

C# Fresher Academy

C# features

C# Features

- Object Oriented Programming: fully support
- Simple: pointers are missing, automatic memory management and garbage collection.
- Modern: C# includes built in support to turn any component into a web service that can be invoked over the internet from any application running on any platform
- Platform Independent: People usually associate C# with Windows, but the language itself is not exclusive of Microsoft's .Net implementation. Examples of existing apps written in C# in the Linux world are [Banshee](#) or [Tomboy](#).



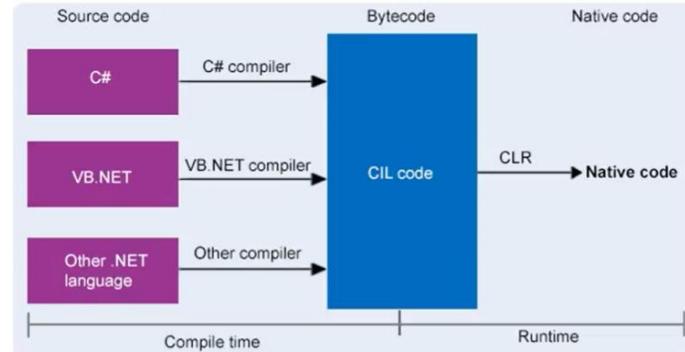
Fresher Academy



- .NET Framework là programming framework cho phép chúng ta xây dựng và triển khai ứng dụng được viết bằng .NET
- Sử dụng .NET Framework có thể giảm thiểu thời gian để viết và triển khai ứng dụng phần mềm
- .NET Framework có 2 phần chính
 - Common language runtime(CLR): được gọi là execution engine chạy trên c#
 - Framework class library(FCL): là một thư viện của class, interface, value type cho phép tương tác với hệ thống. VD:system.io

.NET Framework

CLR



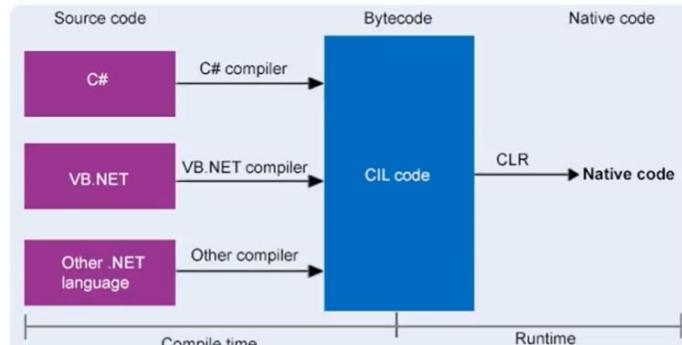
Fresher Academy



C

.NET Framework

CLR



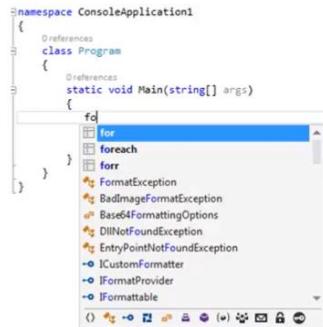
Fresher Academy



- C# codeSnippet là đoạn code dựng sẵn, bạn có thể quickly insert vào trong source code

C# Code Snippet

- Code snippets are ready-made snippets of code you can quickly insert into your code. For example, the **for** code snippet creates an empty **for** loop
- <https://docs.microsoft.com/en-us/visualstudio/ide/visual-csharp-code-snippets>



Fresher Academy



1 6 Nuget

Nuget là cơ chế cho phép lập trình viên có thể tạo, chia sẻ library hữu ích, những library này được gọi là packages

Visual Studio Nuget

Nuget

- Nuget is a mechanism through which developers can create, share, and consume useful libraries of code which called "packages".
- A Nuget package is a single ZIP file with the ".nupkg" extension that contains compiled code (DLLs) and other files related to code.
- 2 main ways install a package:
 - Package Manager UI (Visual Studio)
 - Package Manager Console (Visual Studio): Install-Package <package_name>



17 HelloWorld

- Hàm main sẽ nhận input là một mảng của các string và trả lại giá trị void

```
Console.WriteLine("Hello World!"); //in ra màn hình  
Console.ReadKey(); // để dừng màn hình
```

18 BuildingPart1

- Building có thể giúp các bạn phát hiện ra lỗi trong quá trình compile-time và quá trình run-time.
Trong quá trình compile-time thì chúng ta có thể phát hiện ra lỗi sai về cú pháp, sai về keyword, typematch, còn trong quá trình run-time thì việc building sẽ cho phép chúng ta phát hiện ra lỗi về logic hoặc lỗi về ngữ nghĩa(semantic errors)
- Phương pháp build

Building

Build Methods

Build Method	Benefit
IDE	Create builds immediately and test them in a debugger.
MSBuild command line	Build projects without installing Visual Studio
Team Foundation Build	<ul style="list-style-type: none">Automate your build process as part of a continuous integration/continuous delivery pipeline.Apply automated tests with every build.



Fresher Academy

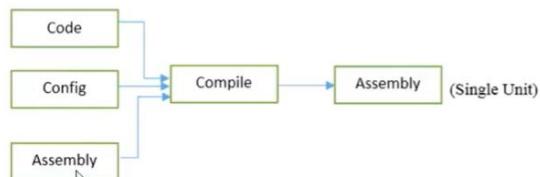


2 3 Assembly And Exe

- Assembly theo định nghĩa của Microsoft thì nó sẽ hình thành ra một đơn vị cơ bản của deployment, hoặc có thể hiểu Assembly là .NET code đã được precompiled và có thể chạy trên CLR
- File exe chỉ là 1 dạng của .NET assembly

Assembly and EXE

- Assembly form the fundamental unit of deployment.



- Assembly is a precompiled .NET code which run by CLR.
- EXE file is a representation of .NET assembly

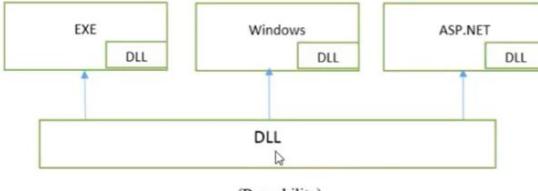


Fresher Academy



2 4 DLL

- DLL(Dynamic Link Library) là một thư viện có thể chứa code và data được sử dụng bởi 1 hoặc nhiều chương trình tại cùng 1 thời điểm



The diagram shows a central horizontal box labeled "DLL". Three arrows point from three separate boxes above it down to the central DLL box. The first box is labeled "EXE" and contains a smaller "DLL" box. The second box is labeled "Windows" and contains a smaller "DLL" box. The third box is labeled "ASP.NET" and contains a smaller "DLL" box. This illustrates that a single DLL can be reused by multiple different applications simultaneously.

Assembly and DLL

- A DLL (Dynamic Link Library) is a library that contains code and data that can be used by more than one program at the same time.

• Assembly can reference as regular DLL

 Fresher Academy 

4 1 Identifiers

- Identifiers là thành phần định danh, là tên để đặt cho class, variable, function hoặc bất cứ item nào chúng ta định nghĩa ra
- Trong C# đặt tên thì nên tuân theo luật sau
 - Tên đặt phải bắt đầu bằng 1 ký tự hoặc dấu gạch dưới_ hoặc dấu @
 - Trong tên không được chứa khoảng trắng, ký tự đặc biệt như ?, -, +!
 - Tên đặt ra không được trùng với keyword
 - Identifiers C# là case-sensitive là phân biệt thường và viết hoa.

Example

```
using System;
namespace BasicOOP
{
    public class Dog
    {
        private int size;

        public void SetSize(int s)
        {
            if (s > 0) size = s;
            else Console.WriteLine("You cannot create a supernatural dog");
        }

        public int GetSize()
        {
            return size;
        }
    }
}
```



Fresher Academy



4.3 Keywords

- Keywords: Từ khoá cũng giống identifiers, đều là những thành phần cơ bản của mọi ngôn ngữ lập trình, từ khoá là tập hợp những từ mà ngôn ngữ lập trình dành sẵn và có ý nghĩa đặc biệt trong 1 chương trình.
- Ý nghĩa của các từ khoá không thể thay đổi và chúng ta không thể sử dụng từ khoá 1 cách trực tiếp như Identifiers trong 1 chương trình. Tuy nhiên nếu muốn sử dụng từ khoá trong 1 chương trình ta sẽ phải chỉ định @trước từ khoá đó

Danh sách từ khoá

Keyword Table

Reserved Keywords						
abstract	as	base	bool	break	byte	case
catch	char	checked	class	const	continue	decimal
default	delegate	do	double	else	enum	event
explicit	extern	false	finally	fixed	float	for
foreach	goto	if	implicit	in	in (generic modifier)	int
interface	internal	is	lock	long	namespace	new
null	object	operator	out	out (generic modifier)	override	params
private	protected	public	readonly	ref	return	sbyte
sealed	short	sizeof	stackalloc	static	string	struct
switch	this	throw	true	try	typeof	uint
ulong	unchecked	unsafe	ushort	using	virtual	void
volatile	while					

 Fresher Academy 

Contextual Keyword

Contextual Keywords							
add	alias	ascending	descending	dynamic	from	get	
global	group	into	join	let 	orderby	partial (type)	
partial (method)	remove	select	set				

 Fresher Academy 

0:02 / 8:37

4 5 NamingConvention

- Quy ước đặt tên là tập hợp các rules do chúng ta đề ra để chọn Identifiers hoặc những thực thể trong source code hoặc trong documentation
- Lợi ích của quy ước đặt tên là:
 - Làm giả được những effort cần thiết để đọc và hiểu code của mọi người trong 1 dự án, tối ưu và giảm thời gian đọc code của chính mình
 - Cho phép tool review code tự động, có thể tập trung vào vấn đề quan trọng của source code hay documentation

Best Practice

do use PascalCasing for class names and method names.

```

1. public class ClientActivity
2. {
3.     public void ClearStatistics()
4.     {
5.         //...
6.     }
7.     public void CalculateStatistics()
8.     {
9.         //...
10.    }
11. }
```

Why: consistent with the Microsoft's .NET Framework and easy to read.





- Sử dụng PascalCasing: là viết hoa chữ cái đầu tiên của mỗi từ cho class và method
- camelCasing: dùng cho method arguments và local variables

Best Practice

do use PascalCasing for class names and method names.

```
1. public class ClientActivity
2. {
3.     public void ClearStatistics()
4.     {
5.         ...
6.     }
7.     public void CalculateStatistics()
8.     {
9.         ...
10.    }
11. }
```

Why: consistent with the Microsoft's .NET Framework and easy to read.

 Fresher Academy 

- không sử dụng Hungarian:

Best Practice

do not use Hungarian notation or any other type identification in identifiers

```
1. // Correct
2. int counter;
3. string name;
4.
5. // Avoid
6. int iCounter;
7. string strName;
```

Why: consistent with the Microsoft's .NET Framework and Visual Studio IDE makes determining types very easy (via tooltips). In general you want to avoid type indicators in any identifier.

 Fresher Academy 

- không sử dụng Screaming Caps cho biến constants hoặc biến readonly (không viết hoa toàn bộ biến)

Best Practice

do not use **Screaming Caps** for constants or readonly variables

```

1. // Correct
2. public static const string ShippingType = "DropShip";
3.
4. // Avoid
5. public static const string SHIPPINGTYPE = "DropShip";

```

Why: consistent with the Microsoft's .NET Framework. Caps grab too much attention.





- tránh việc viết tắt tên biến

Best Practice

avoid using **Abbreviations**. Exceptions: abbreviations commonly used as names, such as **Id**, **Xml**, **Ftp**, **Uri**

```

1. // Correct
2. UserGroup userGroup;
3. Assignment employeeAssignment;
4.
5. // Avoid
6. UserGroup usrGrp;
7. Assignment empAssignment;
8.
9. // Exceptions
10. CustomerId customerId;
11. XmlDocument xmlDoc;
12. FtpHelper ftpHelper;
13. UriPart uriPart;

```

Why: consistent with the Microsoft's .NET Framework and prevents inconsistent abbreviations.





- sử dụng PascalCasing cho viết tắt có nhiều hơn 3 ký tự và sẽ sử dụng uppercase đối với biến có 2 ký tự

Best Practice

do use PascalCasing for abbreviations 3 characters or more (2 chars are both uppercase)

```

1. HtmlHelper htmlHelper;
2. FtpTransfer ftpTransfer;
3. UIControl uiControl;

```

Why: consistent with the Microsoft's .NET Framework. Caps would grab visually too much attention.





- không sử dụng Underscores trong identifiers.
- Nhưng trong private có thể sử dụng Underscores(VD: private Datetime _registrationDate;)

Best Practice

do not use Underscores in identifiers. Exception: you can prefix private static variables with an underscore. ↴

```

1. // Correct
2. public DateTime clientAppointment;
3. public TimeSpan timeLeft;
4.
5. // Avoid
6. public DateTime client_Appointment;
7. public TimeSpan time_Left;
8.
9. // Exception
10. private DateTime _registrationDate;

```

Why: consistent with the Microsoft's .NET Framework and makes code more natural to read (without 'slur'). Also avoids underline stress (inability to see underline).





- Sử dụng implicit type var cho biến local, không nên dùng cho biến kiểu primitive như int,string,double

Best Practice

do

use implicit type `var` for local variable declarations. Exception: primitive types (int, string, double, etc) use predefined names.

```
1. var stream = File.Create(path);
2. var customers = new Dictionary();
3.
4. // Exceptions
5. int index = 100;
6. string timeSheet;
7. bool isCompleted;
```

Why: removes clutter, particularly with complex generic types. Type is easily detected with Visual Studio tooltips.



Fresher Academy



- Nên sử dụng chữ I viết hoa trước interface

Best Practice

do

prefix interfaces with the letter `I`. Interface names are noun (phrases) or adjectives.



```
1. public interface IShape
2. {
3. }
4. public interface IShapeCollection
5. {
6. }
7. public interface IGroupable
8. {
9. }
```

Why: consistent with the Microsoft's .NET Framework.



Fresher Academy



- Sử dụng vertically align curly brackets: align dấu mở đóng ngoặc trên cùng 1 dòng {}

Best Practice

do vertically align curly brackets.

```

1. // Correct
2. class Program
3. {
4.     static void Main(string[] args)
5.     {
6.     }
7. }
```

Why: Microsoft has a different standard, but developers have overwhelmingly preferred vertically aligned brackets.

 Fresher Academy 

- Nên khai báo tất cả biến variable, properties trong 1 class lên trên đầu sau đó mới đến method

Best Practice

do declare all member variables at the top of a class, with static variables at the very top.

```

1. // Correct
2. public class Account
3. {
4.     public static string BankName;
5.     public static decimal Reserves;
6.
7.     public string Number {get; set;}
8.     public DateTime DateOpened {get; set;}
9.     public DateTime DateClosed {get; set;}
10.    public decimal Balance {get; set;}
11.
12.    // Constructor
13.    public Account()
14.    {
15.        // ...
16.    }
17. }
```

Why: generally accepted practice that prevents the need to hunt for variable declarations.

 Fresher Academy 

- Nên sử dụng tên của enums ở dạng singular số ít

Best Practice

do use singular names for enums. Exception: bit field enums.

```

1. // Correct
2. public enum Color
3. {
4.     Red,
5.     Green,
6.     Blue,
7.     Yellow,
8.     Magenta,
9.     Cyan
10.}
11.
12. // Exception
13. [Flags]
14. public enum Dockings
15. {
16.     None = 0,
17.     Top = 1,
18.     Right = 2,
19.     Bottom = 4,
20.     Left = 8
21.}

```

Why: consistent with the Microsoft's .NET Framework and makes the code more natural to read. Plural flags because enum can hold multiple values (using bitwise 'OR').





- Không sử dụng suffix cho enum

Best Practice

do not suffix enum names with Enum

```

1. // Don't
2. public enum CoinEnum
3. {
4.     Penny,
5.     Nickel,
6.     Dime,
7.     Quarter,
8.     Dollar
9. }
10.
11. // Correct
12. public enum Coin
13. {
14.     Penny,
15.     Nickel,
16.     Dime,
17.     Quarter,
18.     Dollar
19. }

```

Why: consistent with the Microsoft's .NET Framework and consistent with prior rule of no type indicators in identifiers.





4.7 DataType

- Data type: là phân loại kiểu dữ liệu và mục đích sử dụng cho từng loại và chia dữ liệu dưới dạng số, dạng chuỗi, thời gian
- C# có 2 categories của data type đó là:

- Value type: lưu những biến sử dụng dưới dạng value type và trả trực tiếp tới vùng nhớ của giá trị được gán tới nó

Value Type Explanation

- A data type is a value type if it holds a data value within its own memory space. It means variables of these data types directly contain their values

int i = 100;

© TutorialsTeacher.com

RAM

i

100

0x239110

C# Fresher Academy

có 2 loại

- Predefined data type: kiểu int, Boolean, float
- User defined data type: kiểu enum

Value Types

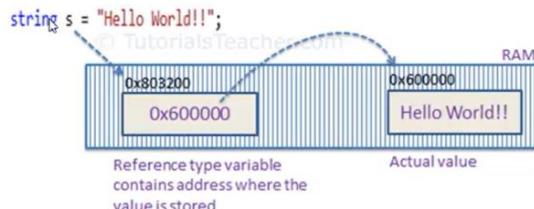
- Numeric types:
 - Integral types (int, uint, long...)
 - Floating-point types (float, double)
 - Decimal
- bool
- enum

C# Fresher Academy

- Reference type: ngược lại với value type là những biến được gán giá trị thì nó sẽ không trả trực tiếp tới vùng nhớ của giá trị mà sẽ chỉ chứa con trỏ và con trỏ sẽ trả tới vùng nhớ đó. Có nghĩa là biến sử dụng reference type thì nó sẽ chứa con trỏ và trả tới giá trị thực.

Reference Type Explanation

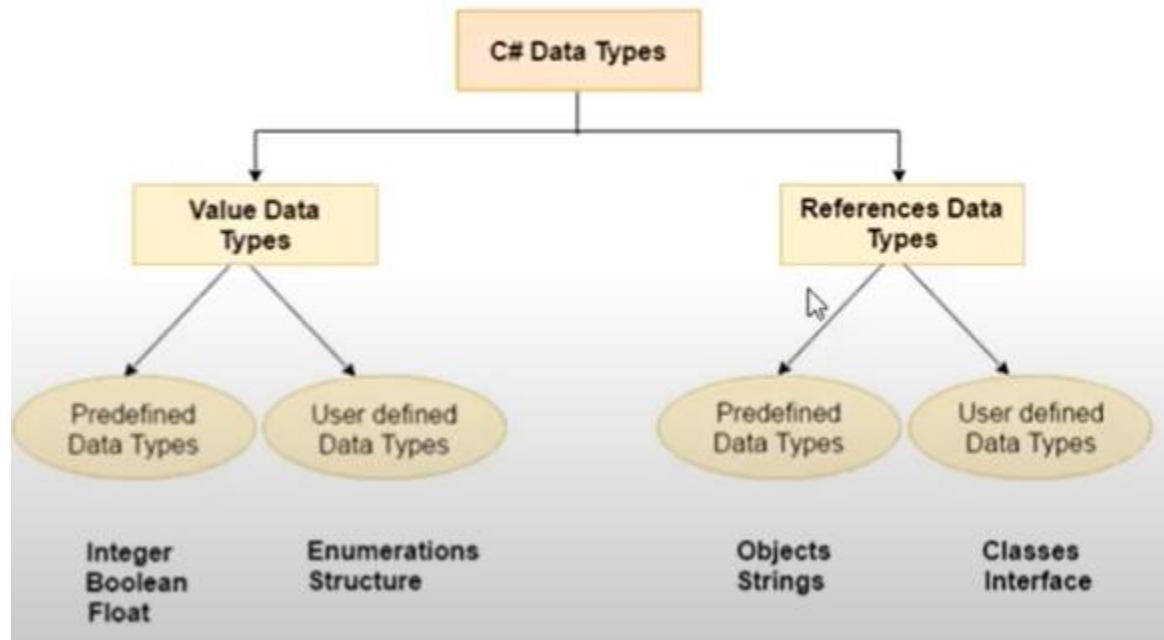
- a reference type doesn't store its value directly. Instead, it stores the address where the value is being stored. In other words, a reference type contains a pointer to another memory location that holds the data



Fresher Academy



- có 2 loại
 - Object string
 - Class interface



4.9 Variable

- Bản thân data có 2 kiểu:
 - Dữ liệu cố định hay còn gọi là const hoặc fix value (dữ liệu ko thay đổi)

- Variable value chính là tên mà chúng ta gọi ra trong data value
- Có 2 kiểu khai báo biến

- <data type> <variable name>;
- <datatype> <variable name> = <value>;

Variable Declaration

- <data type> <variable name>;
- <datatype> <variable name> = <value>;

```
string message;
// value can be assigned after it declared
message = "Hello World!!";

int i, j, k, l = 0;

int amount, num;
```

int i, j,
k,
l = 0;



4 10 VariableInitialization

- Khởi tạo biến trong C#
 - Có thể khai báo biến sử dụng theo 2 cú pháp như sau
 - <data type> <variable name>;
 - <datatype> <variable name> = <value>;
 - Remember that using uninitialized variables in C# is not allowed.
- Giá trị mặc định của references type là null

null

- null is default value for reference-type variables.
- Ordinary value types cannot be null. But C#2.0 has nullable value types

```
// A null string is not the same as an empty string.  
string s = null;  
string t = String.Empty; // Logically the same as ""
```

```
// A value type cannot be null  
// int i = null; // Compiler error!
```

```
// Use a nullable value type instead:  
int? i = null;
```



3 17 VariableScopeVideo

- Phạm(scope) vi của biến(Variable) trong C# là area hoặc region của code mà ở đó chúng ta có thể accept và sử dụng biến đó
- Trong C# có 3 level scope:
 - Class-level scope: những biến được định nghĩa trong 1 class thì nó sẽ available cho tất cả method trong class đấy

Class-Level Scope

- Variables that are defined at the class level are available to any non-static method within the class

```
public class Dog  
{  
    private int size;  
  
    public void SetSize(int s)  
    {  
        if (s > 0) size = s;  
        else Console.WriteLine("You cannot create a supernatural dog");  
    }  
  
    public int GetSize()  
    {  
        return size;  
    }  
}
```



- Method-level scope: những biến được khai báo trong method thì nó sẽ available cho tất cả thành phần còn lại trong method đó

The slide has a wooden background header with the title "Method-Level Scope". Below the header is a list item:

- Variables declared within a method's code block are available for use by any other part of the method, including nested code blocks

The code block shows a C# method named `Demostration`:

```
public void Demostration()
{
    int score;                                // Declared at method-level
    score = 100;                               // Used at method-level

    if (score >= 50)
        Console.WriteLine("Good score : {0}", score); // Used in nested scope
    else
        Console.WriteLine("Poor score : {0}", score); // Used in nested scope
}
```

The slide includes a C# logo, the text "Fresher Academy", and a small icon of books.

- Nested scope : những biến được định nghĩa trong vòng lặp for, while, do-while, loop.v.v thì những biến này sẽ chỉ available trong vòng lặp này và chỉ được accept bởi vòng lặp này. Ở bên ngoài vòng lặp thì sẽ không sử dụng nó, những biến này được gọi là local variables

4 12 VarDynamic

- Kiểu ngầm định cho local variables trong C#
- Local variables là những biến được khai báo và sử dụng trong method, ngoài ra không được sử dụng ở chỗ khác. Khi khai báo 1 biến thì cần chỉ định data type cho biến đó, tuy nhiên C# cho phép

dùng từ khoá var để khai báo biến mà không cần phải chỉ ra tường minh data type cho biến đó.

Implicitly Typed Local Variables

- Local variables can be declared without giving an explicit type. The `var` keyword instructs the compiler to infer the type of the variable from the expression on the right side of the initialization statement

```
// i is compiled as an int
var i = 5;

// s is compiled as a string
var s = "Hello";

// a is compiled as int[]
var a = new[] { 0, 1, 2 };

// expr is compiled as IEnumerable<Customer>
// or perhaps IQueryable<Customer>
var expr =
    from c in customers
    where c.City == "London"
    select c;

// anon is compiled as an anonymous type
var anon = new { Name = "Terry", Age = 34 };

// list is compiled as List<int>
var list = new List<int>();
```



- Chú ý khi sử dụng var
 - Var có thể sử dụng khi biến local variable được khai báo và khởi tạo trong cùng 1 câu lệnh
 - Var không thể dùng để khai báo cho fields trong phạm vi class
 - Không thể sử dụng var để khai báo multiple implicitly-typed
 - Trong hầu hết var được coi là optional và chỉ coi là syntactic(viết cho tiện và nhanh)
- Dynamic không thực hiện việc check kiểu dữ liệu của variable tại thời điểm compile-time mà sẽ chỉ xác định kiểu của biến tại thời điểm runtime

Example

```
dynamic d = "test";
Console.WriteLine(d.GetType());
// Prints "System.String".

d = 100;
Console.WriteLine(d.GetType());
// Prints "System.Int32".
```



Fresher Academy

4 14 StackAndHeap

- Khi khai báo biến variable trong C# thì .NET Framework sẽ allocates vùng nhớ trên ram
- Có 2 loại vùng nhớ: stack và heap
 - Stack : là vùng nhớ trên ram và hoạt động theo cơ chế LIFO(Last in, First out). Mỗi lần khai báo biến thì nó sẽ tự động push vào trong stack, sau đó khi biến fall out scope và function kết thúc thì biến sẽ allocates khỏi stack
 - Heap: là vùng nhớ trên ram tuy nhiên cho phép allocates vùng nhớ 1 cách tự động và sẽ không hoạt động theo cơ chế của stack(LIFO). Ta sẽ tương tác với heap thông qua con trỏ và con trỏ này sẽ chỉ tới vùng nhớ trên heap

Code Sample

```
public void Method1()
{
    // Line 1
    int i=4;

    // Line 2
    int y=2;

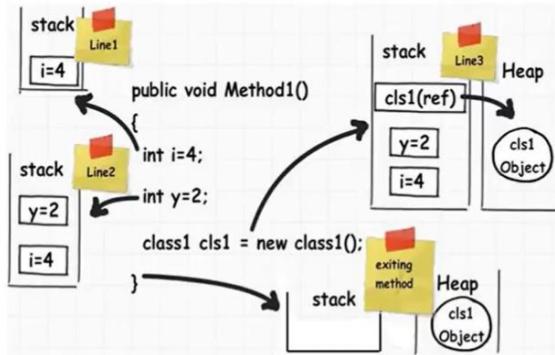
    //Line 3
    class1 cls1 = new class1();
}
```



Fresher Academy



Explanation



Fresher Academy



- Khi một function kết thúc thì stack sẽ tự động empty còn heap thì nó vẫn chứa dữ liệu
- Đặc điểm của stack
 - Hoạt động nhanh
 - Không cần phải explicitly de-allocate biến
 - Quản lý bởi CPU nên ram sẽ không bị phân mảnh
- Đặc điểm của Heap
 - Không giới hạn về kích thước bộ nhớ
 - Hoạt động chậm hơn stack
 - Bộ nhớ heap sẽ bị fragmented theo thời gian

OutvsRefVideo

- Về cơ bản out và ref sẽ cho phép chúng ta pass 1 biến dạng references với một method
- Khác biệt giữa ref và out
 - Ref không cần khởi tạo giá trị khi truyền vào
 - Out cần khởi tạo giá trị cho method

4 15 BoxingUnboxing

- Boxing là quá trình convert value type trở thành data type object (data type obj là references type). Khi CLR command language runtime boxes wraps giá trị đó vào trong system.obj sau đó lưu giá trị đó lên heap
- Unboxing thì ngược lại thì nó sẽ extracts value type từ obj

- Boxing là ngầm định, unboxing thì là tường minh.

Boxing Sample

```
int i = 123;
// Boxing copies the value of i into object o.
object o = i;
```

On the stack

On the heap

Boxing Conversion

Fresher Academy

Unboxing Sample

```
int i = 123;      // a value type
object o = i;     // boxing
int j = (int)o;   // unboxing
```

On the stack

On the heap

Unboxing Conversion

Fresher Academy

- Lợi ích của boxing và unboxing :
 - Cho phép chúng ta có một cái nhìn thống nhất về data type của C#, chúng ta có thể treated value giống references type
 - Boxing có thể đẩy kiểu int vào trong collection array list

4 16 GarbageCollection

- Mỗi lần khai báo biến trong C# thì tự động .NET Framework đẩy biến vào trong stack, với value type thì nó trỏ trực tiếp vào giá trị, còn giá trị của references type được lưu trên heap. Biến trên stack trỏ đến vùng nhớ trên heap
- Stack và heap đều là bộ nhớ trên ram và ram thì không phải vô tận, chính vì vậy nếu chúng ta khai báo quá nhiều ref type thì heap sẽ tăng size đến giới hạn ram và chương trình sẽ không chạy
- .NET Framework có .NET garbage collector sẽ quản lý vùng nhớ cho chúng ta/ Garbage collector sẽ làm nhiệm vụ cấp phát và giải phóng bộ nhớ cho ứng dụng
 - Mỗi lần tạo object thì CLR sẽ allocate bộ nhớ cho obj để quản lý trên heap
 - Check obj trên heap có còn được sử dụng hay không , sau một thời gian dài mà không được sử dụng thì nó sẽ xoá
- Garbage collector sẽ sử dụng khái niệm generation có nghĩa là vùng nhớ heap sẽ được chia thành các vùng generation long-live và short-lived object(obj có vòng đời dài và ngắn)
 - Generation 0: gọi là youngest generation và nó chỉ chứa obj có vòng đời ngắn. Ví dụ về short-lived obj là các biến temporary, chúng ta có một function và khai báo những biến trong function đó sẽ gọi là temporary variable
 - Generation 1: chứa short-lived obj và nó hoạt động giống như buffer giữa short và long-lived obj. Những obj sống sót qua generation 1 thì sẽ chuyển qua generation 2
 - Generation 2 : chỉ chứa long-lived obj(obj có vòng đời dài) và đây là generation cuối
- Lợi ích của garbage collector
 - Không cần lo lắng về cấp phát vùng nhớ và lấy lại vùng nhớ, chỉ đơn giản viết ra obj thì tự động garbage collector sẽ dọn dẹp vùng nhớ cho chúng ta và chỉ cần tập trung vào nghiệp vụ
 - Garbage collector sử dụng series thuật toán để allocated và lưu một cách tốt nhất.

6 0 Introduction Video

- Operators: toán tử , kí hiệu để thực thi hoạt động trên những toán hạng. Các toán hạng có thể là biến, có thể là const
- Ví dụ: 2+3 thì 2 và 3 là toán hạng còn + là toán tử

Section Intro

- Operators are symbols that are used to perform operations on operands. Operands may be variables and/or constants.
- in `2+3`, `+` is an operator that is used to carry out addition operation, while `2` and `3` are operands
- Operators in this section
 - Arithmetic
 - Assignment
 - Relational
 - Conditional
 - Logical
 - instanceof Comparison



Fresher Academy



6 1 Arimetic Video

Toán tử toán học được sử dụng để thực thi những hoạt động toán học giữa những toán hạng khác nhau. Những hoạt động này bao gồm `+-*/`

Arithmetic Operator

- Arithmetic operators are used to perform arithmetic operations such as addition, subtraction, multiplication, division, ..

Operator	Description	Example
<code>+</code>	Adds two operands	<code>A + B = 30</code>
<code>-</code>	Subtracts second operand from the first	<code>A - B = -10</code>
<code>*</code>	Multiplies both operands	<code>A * B = 200</code>
<code>/</code>	Divides numerator by de-numerator	<code>B / A = 2</code>
<code>%</code>	Modulus Operator and remainder of after an integer division	<code>B % A = 0</code>
<code>++</code>	Increment operator increases integer value by one	<code>A++ = 11</code>
<code>--</code>	Decrement operator decreases integer value by one	<code>A-- = 9</code>



Fresher Academy



6 2 Assigment Video

- Assignment operator: toán tử gán sẽ gán giá trị từ toán hạng bên tay phải sang toán hạng bên tay trái
- Toán tử $=$ sẽ gán giá trị từ toán hạng bên phải sang bên trái ví dụ : $C = A + B$ thì lúc này nó sẽ gán $A + B$ cho C
- Toán tử $+=$ nó sẽ cộng toán hạng bên trái và bên phải ra giá trị bao nhiêu thì nó sẽ gán lại vào trong toán hạng bên trái. Ví dụ: $C += A$ thì nó sẽ $+ C$ với A ra giá trị bao nhiêu thì nó sẽ gán lại vào C
- Toán tử $-=$ ngược lại với $+=$ thì nó sẽ trừ toán hạng bên trái cho bên phải sau đó được giá trị bao nhiêu thì sẽ gán ngược lại bên trái. Ví dụ $C -= A$ thì nó sẽ ngược lại $C - A$ bằng bao nhiêu thì gán cho C
- Toán tử $*=$ sẽ nhân 2 toán hạng với nhau và so sánh
- Toán tử $/=$ lấy phần dư của toán hạng bên trái và chia cho bên phải được bao nhiêu thì gán lại vào bên trái

Assignment Operator

- Assignment operators store a value in the object designated by the left operand

Operator	Description	Example
$=$	Simple assignment operator, Assigns values from right side operands to left side operand	$C = A + B$ assigns value of $A + B$ into C
$+=$	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	$C += A$ is equivalent to $C = C + A$
$-=$	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	$C -= A$ is equivalent to $C = C - A$
$*=$	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	$C *= A$ is equivalent to $C = C * A$
$/=$	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	$C /= A$ is equivalent to $C = C / A$
$\%=$	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	$C \%= A$ is equivalent to $C = C \% A$


C#


6 3 Conditional Video

- Conditional operator trong C# được kí hiệu là $(?:)$ và nó sẽ trả lại 1 trong 2 giá trị phụ thuộc vào biểu thức boolean
- Cú pháp như sau:

```
condition ? first_expression : second_expression;
```

- Đầu tiên có biểu thức điều kiện(condition), khi mà condition là true thì nó sẽ trả về first_expression còn nếu trả về false thì sẽ trả về second_expression
- Lưu ý là first_expression và second_expression đều phải cùng kiểu dữ liệu hoặc ngầm định convert giữa 2 loại

6.4 Comparison Video

- So sánh trong C# có 2 loại:
 - So sánh bằng (value equality) sẽ quyết định xem là giá trị chứa trong 2 biến có bằng nhau hay không
 - So sánh bằng thứ 2 là reference equality nó sẽ quyết định xem 2 biến nó có trỏ đến 1 vùng nhớ hay không
- Về reference Equality chúng ta có thể sử dụng phương thức gọi là object.ReferenceEquals và phương thức này trả về true hoặc false
 - True có nghĩa là 2 biến cùng trỏ tới 1 vùng nhớ
 - False 2 biến trỏ tới 2 vùng nhớ khác nhau

The slide has a wooden background. The title 'Reference Equality' is at the top. Below it, the text 'Object.ReferenceEquals' is displayed. Two code snippets are shown in a dark box:

```
Dog bruno = new Dog("Bruno", "Bull");
Dog bruno1 = new Dog("Bruno", "Bull");
Console.WriteLine(Object.ReferenceEquals(bruno, bruno1));
// False
```



```
Dog bruno2 = bruno;
Console.WriteLine(Object.ReferenceEquals(bruno, bruno2));
// True
```

At the bottom left is the C# logo. In the center is the text 'Fresher Academy'. At the bottom right is an icon of an open book.

- Value Equality

Value Equality

```

int v1 = 10;
int v2 = 10;
Console.WriteLine(v1 == v2);
// True

```

```

Dog bruno = new Dog("Bruno", "Bull");
Dog bruno1 = new Dog("Bruno", "Bull");
Console.WriteLine(Object.ReferenceEquals(bruno, bruno1));
// False

```

```

    public override bool Equals(object obj)
{
    var other = obj as Dog;
    if (other == null)
        return false;
    if (Name != other.Name || Breed != other.Breed)
        return false;
    return true;
}

```

```

Console.WriteLine(bruno.Equals(bruno1));
// True

```





6 5 Logical Video

- Toán tử Logic thực thi hoạt động logic giữa những biến với nhau, hoạt động logic bao gồm and, or và nó sẽ trả lại giá trị true, false. Những toán tử logic này hay được sử dụng khi chúng ta sử dụng decision making if else và sử dụng loop
 - Toán tử and có kí hiệu &&: nó sẽ chỉ trả lại giá trị true khi 2 toán hạng đều là true
 - Toán tử or kí hiệu ||: toán tử này sẽ chỉ trả về true nếu 1 trong 2 toán hạng là true
 - Toán tử not kí hiệu !:

8 1 IFElse Video

- Trong C# if là câu lệnh điều kiện có nghĩa là nếu biểu thức trong câu lệnh if đúng thì chúng ta sẽ thực thi 1 số câu lệnh. Còn trong trường hợp else thì chúng ta sẽ thi một số câu lệnh khác
- Trong C# có thể follow bởi 0 hoặc 1 câu lệnh else(optional: có thể có hoặc không)

Syntax

```
if(boolean_expression) {  
    /* statement(s) will execute if the boolean expression is true */  
}  
else {  
    /* statement(s) will execute if the boolean expression is false */  
}
```

- **Expression** is something return value. $x > 5$ or $x + 2$
- **Statement** is code that does something.
`If (x > 5) Console.WriteLine("greater than 5");`



Fresher Academy



- Expression: biểu thức là thứ gì đấy trả lại giá trị
- Statement là những đoạn code để làm một việc gì đấy sẽ không trả lại giá trị
- Else if...else Statement
 - Else if luôn đứng sau if và đứng trước else

If...else if...else Statement

- **if** statement can be followed by an optional **else if...else** statement
 - An if can have zero or one else's and it must come after any else if's
 - An if can have zero to many else if's and they must come before the else
 - Once an else if succeeds, none of the remaining else if's or else's will be tested

```
if(boolean_expression 1) {  
    /* Executes when the boolean expression 1 is true */  
}  
else if( boolean_expression 2) {  
    /* Executes when the boolean expression 2 is true */  
}  
else if( boolean_expression 3) {  
    /* Executes when the boolean expression 3 is true */  
} else {  
    /* executes when the none of the above condition is true */  
}
```



- Nested if Statement: Tức là sử dụng if else lồng nhau trong C#, có nghĩa chúng ta có thể sử dụng câu lệnh if else nằm trong câu lệnh if else khác

Nested if Statement

- It is always legal in C# to **nest** if-else statements, which means you can use one if or else if statement inside another if or else if statement(s)

```
if( boolean_expression 1) {  
    /* Executes when the boolean expression 1 is true */  
    if(boolean_expression 2) {  
        /* Executes when the boolean expression 2 is true */  
    }  
}
```



Fresher Academy



8 3 Switch1Video

- Constant pattern: một lệnh switch có thể cho phép chúng ta so sánh một biến với một danh sách các giá trị và mỗi giá trị này gọi là một case(trường hợp)

```
switch(expression) {  
    case constant-expression :  
        statement(s);  
        break; /* optional */  
    case constant-expression :  
        statement(s);  
        break; /* optional */  
  
    /* you can have any number of case statements */  
    default : /* Optional */  
        statement(s);  
}
```

- Lưu ý expression ở đây là enum hoặc kiểu số
- Break: dừng điều kiện khi đã thỏa mãn và thoát ra(optional)

8 5 SwitchNewVideo

- Constant pattern: giá trị trong case luôn là dạng số hoặc enum

Challenge

Requirement

- Write a program check age of user:
 - If age is in 6, 7, 8, 9, 10 display "Elementary" on the screen.
 - If age is in 11, 12, 13, 14 display "Middle School" on the screen.
 - If age is in 15, 16, 17 display "High School" on the screen.
 - Otherwise, display "Out of Range".

```
default:  
    Console.WriteLine("Out Of Range");  
    break;
```

```
int age = 12;  
switch(age)  
{  
    case 6:  
    case 7:  
    case 8:  
    case 9:  
    case 10:  
        Console.WriteLine("Elementary");  
        break;  
  
    case 11:  
    case 12:  
    case 13:  
    case 14:  
        Console.WriteLine("Middle School");  
        break;  
  
    case 15:  
    case 16:  
    case 17:  
        Console.WriteLine("High School");  
        break;  
  
    default:  
        Console.WriteLine("Out of Range");  
        break;
```



Fresher Academy



Viết gọn hơn với new Approach trong C#:

Compare 2 Solutions

```
int age = 12;  
switch(age)  
{  
    case 6:  
    case 7:  
    case 8:  
    case 9:  
    case 10:  
        Console.WriteLine("Elementary");  
        break;  
    case 11:  
    case 12:  
    case 13:  
    case 14:  
        Console.WriteLine("Middle School");  
        break;  
    case 15:  
    case 16:  
    case 17:  
        Console.WriteLine("High School");  
        break;  
    default:  
        Console.WriteLine("Out of Range");  
        break;  
}
```

```
int age = 12;  
switch(age)  
{  
    case int n when (6 <= n && n < 11):  
        Console.WriteLine("Elementary");  
        break;  
    case int n when (11 <= n && n < 15):  
        Console.WriteLine("Middle School");  
        break;  
    case int n when (15 <= n && n < 18):  
        Console.WriteLine("High School");  
        break;  
    default:  
        Console.WriteLine("Out of Range");  
        break;  
}
```



Fresher Academy



- Type pattern: trong switch thì expression luôn phải là dạng số hoặc enum

Type pattern

```
static void Main(string[] args)
{
    int[] values = { 2, 4, 6, 8 };

    var names = new List<String>();
    names.Add("An");
    names.Add("Binh");
    names.Add("Cuong");

    DisplayCollectionInformation(values);
    DisplayCollectionInformation(names);
    Console.ReadKey();
}

// references
private static void DisplayCollectionInformation(object collection)
{
    switch (collection) {
        case Array arr:
            Console.WriteLine("This is an array and has " + arr.Length);
            break;
        case IList list:
            Console.WriteLine("This is a list and has " + list.Count);
            break;
    }
}
```

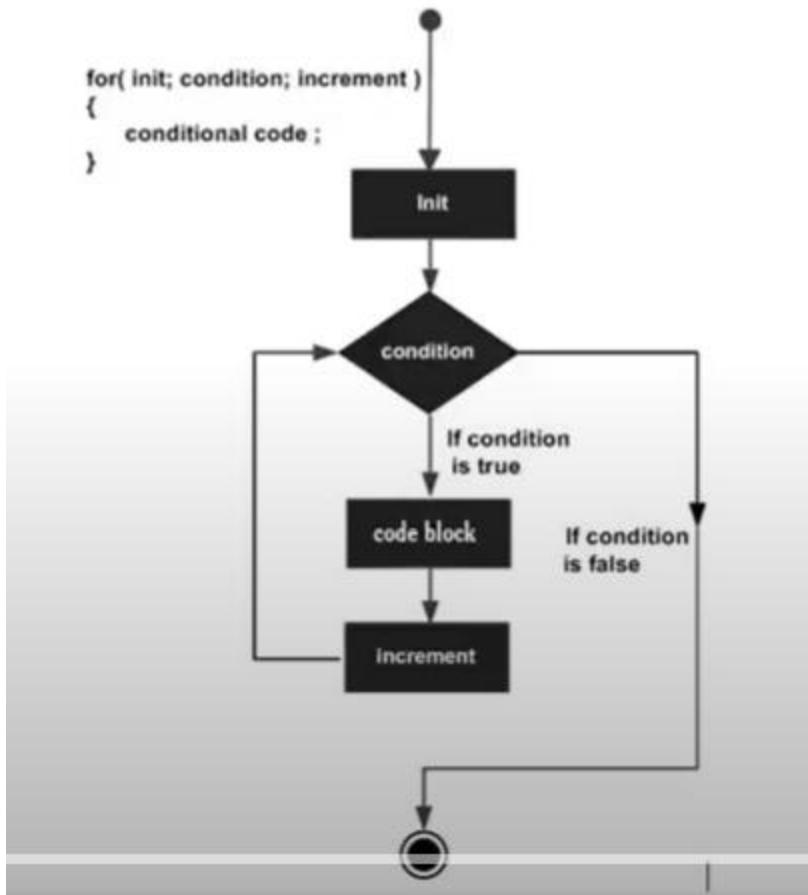


Fresher Academy



8 7 ForLoopVideo

- Trong C# có thể dùng câu lệnh for để chạy khối lệnh lặp đi lặp lại cho đến khi một expression bị đánh giá là false hoặc chúng ta tường minh thoát ra khỏi vòng for đó
- Cú pháp



Trong đó:

- init cho phép chúng ta có thể khai báo và khởi tạo biến để control và kiểm soát vòng lặp, init có thể là optional(dùng hoặc không)
- condition: trả về true hoặc false. Trong trường hợp trả về true thì nó sẽ thực thi khối lệnh trong phần thân lệnh for và sau khi thực thi xong thì nó sẽ chạy đến increment(tăng giá trị của biến điều khiển vòng loop này lên và sau đó nó đánh giá lại condition, trong trường hợp condition trả về true thì sẽ tiếp tục chạy, còn nếu false thì nó sẽ interminate)(increment là optional)
- dùng vòng for dùng break;
- trong C# sử dụng foreach giúp chúng ta viết code dễ dàng hơn và đọc nó dễ hơn. Foreach sẽ làm nhiệm vụ chạy qua từng item trong mảng hoặc collection sẽ kiểm tra có còn phần tử nào nằm trong collection mà nó chưa chạy qua hay không, trong trường hợp vẫn còn phần tử như vậy thì nó sẽ gán phần tử tiếp theo của collection vào biến local và thực thi lệnh trong phần thân foreach. Trong trường hợp không còn element nào trong collection thì nó sẽ thoát ra khỏi lệnh foreach

8 9 WhileDoWhileVideo

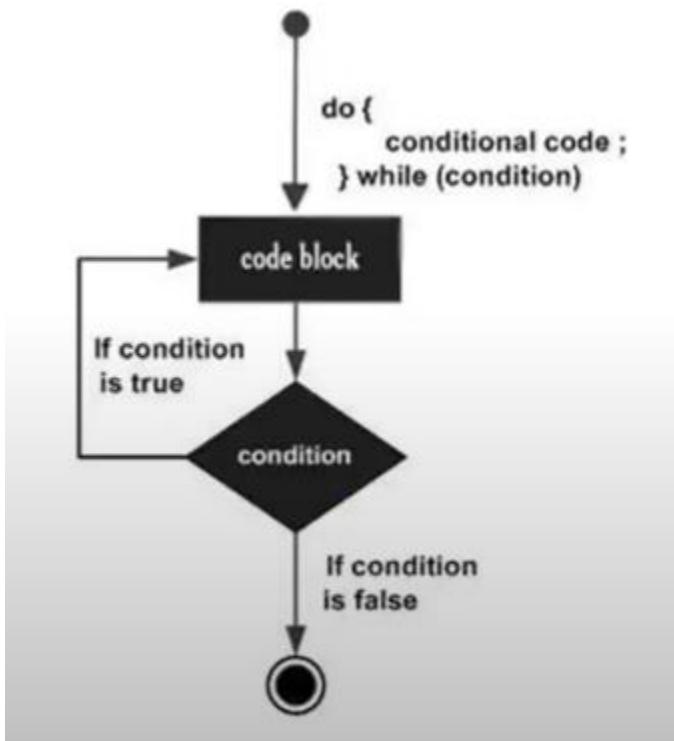
- vòng lặp while sẽ cho phép chúng ta có thể thực thi 1 câu lệnh hoặc 1 khối câu lệnh và chừng nào biểu thức chúng ta đưa ra đúng thì cú pháp của câu lệnh while sẽ là

```

while( condition )
{
    conditional code ;
}

```

- khối lệnh sẽ được thực thi khi condition này trả về true
- do while loop cho phép chúng ta chạy khối câu lệnh cho đến khi biểu thức chúng ta đưa ra là false



- thực thi code block và kiểm tra điều kiện condition(nếu true tiếp tục thực thi) nếu false thì dừng lại

8 12 BreakContinueVideo

- break sẽ được dùng trong vòng lặp để terminate và dùng cho for,foreach, while, do..while

Break

- **break** statement in loop is used to terminate the current loop iteration. It is used in for, foreach, while, do..while

```
class BreakTest
{
    static void Main()
    {
        for (int i = 1; i <= 100; i++)
        {
            if (i > 5)
            {
                break;
            }
            Console.WriteLine(i);
        }
    }
}
```

* Output:
1
2
3
4



Ví dụ break Nested Loop

Fresher Academy



≡ 8 12 BreakContinueVideo

Example Break Nested Loop

```
/*
 * Output:
 * num = 0
 *
 * num = 1
 * a
 * num = 2
 * a b
 * num = 3
 * a b c
 * num = 4
 * a b c d
 * num = 5
 * a b c d e
 * num = 6
 * a b c d e f
 * num = 7
 * a b c d e f g
 * num = 8
 * a b c d e f g h
 * num = 9
 * a b c d e f g h i
 */

int[] numbers = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
char[] letters = { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j' };

// Outer loop
for (int x = 0; x < numbers.Length; x++)
{
    Console.WriteLine("num = {0}", numbers[x]);

    // Inner loop
    for (int y = 0; y < letters.Length; y++)
    {
        if (y == x)
        {
            // Return control to outer loop
            break;
        }
        Console.Write(" {0} ", letters[y]);
    }
    Console.WriteLine();
}
```



0:49 / 3:12

Fresher Academy



- continue dùng để skip việc thực thi current iteration, nó sẽ không break cả loop

Continue

- `continue` skip the execution of current iteration only and it does not break the loop. It passes the control to the next iteration.

```
class ContinueTest
{
    static void Main()
    {
        for(int i = 1; i <= 10; i++)
        {
            if (i < 9)
            {
                continue;
            }
            Console.WriteLine(i);
        }

        // Keep the console open in debug mode.
        Console.WriteLine("Press any key to exit.");
        Console.ReadKey();
    }
}

/*
Output:
9
10
*/
```



Fresher Academy



- break kết thúc current loop, còn continue sẽ skip thực thi current iteration

7 0 Introduction video

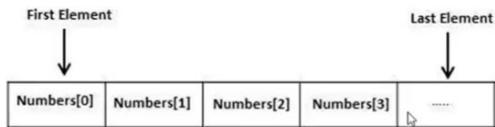
- array: mảng là cấu trúc dữ liệu cho phép chúng ta tổ chức lưu trữ dữ liệu theo một cách đặc biệt để có thể truy nhập đến, sửa đổi một cách tốt nhất. Cấu trúc dữ liệu được xem như là một tập hợp của data value, mối quan hệ giữa chúng và function có thể áp dụng lên trên dữ liệu
- mảng trong C# gồm những phần tử thông thường có cùng kiểu và được tổ chức theo thứ tự cụ thể, chúng ta có thể lấy ra phần tử trong mảng bằng cách sử dụng index

7 1 DeclaringArray video

- mảng trong C# là tập hợp tuần tự của các phần tử có cùng kiểu và có kích thước cố định
- khi khai báo mảng thì chúng ta sẽ tạo ra rất nhiều vùng nhớ liên kết với nhau trên stack. Vị trí các phần tử trong mảng gọi là index

Array Revisit

- An array stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type stored at contiguous memory locations



Fresher Academy



Datatype[] arrayName;

7.2 Initialize an array

- khai báo mảng nhưng chưa khởi tạo mảng trong vùng nhớ
- mảng là kiểu ref type nên cần sử dụng từ khoá new để có thể tạo ra instance cho mảng
- các cách gán giá trị tới mảng
 - gán giá trị tới từng element của mảng đó
 - gán giá trị cho mảng ngay thời điểm khai báo mảng đó
 - có thể vừa khởi tạo mảng vừa gán giá trị cho nó
 - loại bỏ kích thước của array

Assigning Values to an Array

- You can assign values to individual array elements, by using the index number

```
double[] balance = new double[10];
balance[0] = 4500.0;
```
- You can assign values to the array at the time of declaration

```
double[] balance = { 2340.0, 4523.69, 3421.0};
```
- You can also create and initialize an array

```
int [] marks = new int[5] { 99, 98, 92, 97, 95};
```
- You may also omit the size of the array

```
int [] marks = new int[] { 99, 98, 92, 97, 95};
```



Fresher Academy



7 3 MultiDimentionalArray video

- C# cho phép chúng ta tạo ra mảng nhiều chiều chẳng hạn bạn muốn khai báo ra mảng 2 chiều kiểu chuỗi chúng ta có thể viết như sau:

Multidimensional Arrays

- C# allows multidimensional arrays. You can declare a 2-dimensional array of strings as:

```
string [,] names;
```

- a 3-dimensional array of int variables as

```
int [ , , ] m;
```

(source: tutorialspoint)



Fresher Academy



- Mảng 2 chiều

Two-Dimensional Array

- A 2-dimensional array can be thought of as a table, which has x number of rows and y number of columns

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

- Every element in the array a is identified by an element name of the form a[i , j] with i: row, j: column



Fresher Academy



- Khởi tạo và access trong mảng 2 chiều
 - Khi khởi tạo mảng 2 chiều thì dùng từ khóa new

Initializing and Accessing

- Multidimensional arrays may be initialized by specifying bracketed values for each row

```
int [,] a = new int [3,4] {  
    {0, 1, 2, 3} , /* initializers for row indexed by 0 */  
    {4, 5, 6, 7} , /* initializers for row indexed by 1 */  
    {8, 9, 10, 11} /* initializers for row indexed by 2 */  
};
```

- Accessing

```
int val = a[2,3];
```



Fresher Academy



7 4 Accessing Video

- Có thể lấy 1 phần tử trong mảng bằng cách sử dụng index, tức là sử dụng vị trí phần tử trong mảng. Lưu ý vị trí các phần tử sẽ bắt đầu từ 0 trở đi chứ không phải là từ 1

Accessing Array Elements

```
double salary = balance[9];
```

- An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array

(source: tutorialspoint)



5.1 DateTime

- GMT(Greenwich Mean Time) hoặc UT(Universal Time)
- 1 ngày trong GMT được đo bằng thời gian quay của trái đất xung quanh trục
- Giờ UTC(Coordinated Universal Time or Temps Universel Coordonné) : sử dụng đồng hồ lượng tử đặt trên thế giới.
- Múi giờ: TimeZone
- Chuẩn thời gian ISO 8601

DateTime in General

- GMT (Greenwich Mean Time) or UT (Universal Time)
- UTC (Coordinated Universal Time or Temps Universel Coordonné)
- TimeZone: defines LocalDateTime
 - . AT....Atlantic Time.....GMT-4 -> UTC-4
 - . GMT....Greenwich Mean Time.....GMT 0 -> [UTC±00](#)
 - . ICT....Indochina Time.....GMT +7 -> UTC+7
- ISO 8601
 - 2030-02-26T05:36:09+00:00
 - 2030-02-26T05:36:09Z
 - 2030-02-26T12:36:09+07:00
- Unix (Epoch time): defined as the number of seconds from 00:00:00 on 1 January 1970.
<https://www.epochconverter.com/> 1898318538 -> 2030-02-26T06:42:18Z. 1 day = 86400



Fresher Academy



- Date Time trong C#
 - DateTime chứa 2 thành phần đó là Date: ngày tháng năm, Time: giờ phút giây
 - Thời gian trong C# định nghĩa từ 12:00AM 01-01-0001 đến 31-12-9999
- Cách khởi tạo DateTime trong C#
 - DateTime(Int32, Int32, Int32): lần lượt Int32 là ngày tháng năm
 - Ví dụ: new DateTime(26-02-2032)

DateTime in C#

- DateTime consist Date and Time. Date in format: **YYYYMMDD**. Time as **HHMMSS**. Value of DateTime is between 0001-01-01 12:00:00AM to 9999-12-31 23:59:59PM
- Initialize DateTime
 - `DateTime(Int32, Int32, Int32)`. For ex: New DateTime(2030, 02, 26) -> 2030-02-26 12:00:00 AM
 - `DateTime(Int32, Int32, Int32, Int32, Int32, Int32)`. For ex: New DateTime(2030, 02, 26, 13, 30, 15) -> 2030-02-26 1:30:15 PM
 - `DateTime(Int32, Int32, Int32, Int32, Int32, Int32, DateTimeKind)`. 3 types: UTC, Local, Undefined



Fresher Academy



- Common Properties

- DayOfWeek : trả lại tên ngày trong tuần
- DayOfYear: trả lại ngày trong năm
- TimeOfDay: trả lại thành phần thời gian trong kiểu DateTime
- ToDay: trả lại ngày tại thời điểm gọi thuộc tính today
- Now : trả lại thời gian hiện tại
- UtcNow: trả lại thời gian UTC chứ không phải local

5.3 DateTimeFormat Video

- DateTime format dùng để định nghĩa để hiển thị text cho giá trị date và giá trị time
- CultureInfo trả lại thông tin của specific culture. Trong DateTime khi chúng ta sử dụng CultureInfo khác nhau thì nó sẽ hiển thị ra định dạng khác nhau

The screenshot shows a video player interface with a slide titled "DateTime Format". The slide contains the following text:

- We use date time format to define the text representation of a date and time value.
- CultureInfo represents information about a specific culture. In DateTime it show different format, 26 Feb 2030
 - en-US: 2/26/2030
 - en-GB, fr-FR, vi-VN: 26/02/2030.
 - ja-JP: 2030/02/26

At the bottom of the slide, there is a logo for "Fresher Academy" and some social media icons. The video player has a black border.

Format DateTime

DateTime Format (2)

new DateTime(2030, 02, 26, 12, 30, 50);

Format	Description	Sample
"d"	Short date pattern.	en-US: 2/26/2030 en-GB, fr-FR, vi-VN: 26/02/2030
"D"	Long date pattern.	en-US: 26 February 2030 vi-VN: 26 Tháng 2 2030
"F"	Full date/time pattern (short time).	en-US: Tuesday, February 26, 2030 12:30 PM vi-VN: 26 Tháng 2 2030 12:30 CH
"F"	Full date/time pattern (long time).	en-US: Tuesday, February 26, 2030 12:30:50 PM vi-VN: 26 Tháng 2 2030 12:30:50 CH
"t"	Short time pattern.	en-US: 12:30 PM vi-VN: 12:30 CH
"T"	Long time pattern.	en-US: 12:30:50 PM vi-VN: 12:30:50 CH


FRESHER ACADEMY


- Convert chuỗi string sang DateTime
 - Phương thức TryParse: convert string thành kiểu datetime và parameter cuối cùng trong phương thức tryparse sẽ out ra kiểu datetime, nếu parse thành công thì nó sẽ trả về true còn nếu không thành công thì trả về false

```

DateTime tmp;
String str1 = "02/26/2030";
DateTime.TryParse(str1, CultureInfo.InvariantCulture, DateTimeStyles.None, out tmp);
/* Output is 02/26/2030 12:00:00 AM*/

```

- Phương thức TryParseExact: hoạt động tương tự như TryParse nhưng nó cho phép chúng ta truyền format mà chúng ta muốn parse ra

```

DateTime tmp2;
String str2 = "20302602 05:30:20";
DateTime.TryParseExact(str2, "yyyyddMM hh:mm:ss", CultureInfo.InvariantCulture, DateTimeStyles.None, out tmp2);
/* Output is 02/26/2030 05:30:20 AM*/

```

5 5 TimeSpanAndDateTimeMethods

- TimeSpan: biểu thị độ dài của thời gian
- Khởi tạo TimeSpan
 - TimeSpan truyền vào giờ, phút, giây
 - TimeSpan truyền vào ngày, giờ, phút, giây
 - TimeSpan truyền vào day, giờ, phút, giây, mili giây
 - TimeSpan(int hours,int minutes,int seconds)
 - TimeSpan(int days, int hours, int minutes, int seconds)
 - TimeSpan(int days, int hours, int minutes, int seconds, int milliseconds)

- Properties
 - Ngày,gìờ,phút,giây, mili giây
- Phương thức(Methods)
 - Add
 - Compare 2 TimeSpan với nhau
- DateTime Methods
 - Add: có thể cộng khoảng thời gian vào biến thời gian và chúng ta khai báo và phương thức sẽ trả về kiểu DateTime

```
DateTime future = new DateTime(2030, 02, 26);
Console.WriteLine(future.Add(new TimeSpan(1, 0, 90, 0)));
/* Output is: 2030-02-27 01:30:00 AM */
```

- TimeSpan Subtract(DateTime): trừ 2 khoảng thời gian cho nhau

Date Time Methods (1)

TimeSpan Subtract(Datetime)

```
DateTime day1 = new DateTime(2030, 01, 29, 12, 0, 0);
DateTime day2 = new DateTime(2030, 01, 23, 12, 0, 0);
DateTime day3 = new DateTime(2030, 01, 23, 13, 0, 0);

Console.WriteLine(day1.Subtract(day2));
/* Output is: 6.00:00:00 */

Console.WriteLine(day1.Subtract(day3));
/* Output is: 5.23:00:00 */
```



Fresher Academy



- DateTime method có thể cộng thêm năm, thêm tháng, thêm ngày, giờ, phút, giây, mili giây vào trong 1 datetime

DateTime Methods (2)

- AddYears(Int32), AddMonths(Int32), AddDays(Double), AddHours(Double), AddMinutes(Double), AddSeconds(Double), AddMilliseconds(Double)

```
future = new DateTime(2030, 01, 29);
Console.WriteLine(future.AddYears(1));
/* Output is: 2031-01-29 12:00:00 AM*/

Console.WriteLine(future.AddMonths(1));
/* Output is: 2030-02-28 12:00:00 AM*/

Console.WriteLine(future.AddDays(30));
/* Output is: 2030-02-28 12:00:00 AM*/
```



Fresher Academy



5.7 String Video

- What is literal

Literal là giá trị chúng ta sẽ hard-coded trực tiếp vào trong source code

Ví dụ:

- String x = "Hello World"; // Hello World là literal
- Int y = 2; // 2 là literal
 - String x = "Hello World"; // Hello World is literal
 - Int y = 2; // So is 2, but not y
 - Int z = y + 4; // y and z are not literals, but 4 is
 - Int a = 1 + 2; // 1 + 2 is not a literal, it is an expression, but 1 and 2 is.

- String là tập hợp của character

- Đôi khi trong C# chúng ta viết string S là viết thường hoặc viết hoa

- Object của string là immutable: có nghĩa là không đổi

String

- A string is one of the most important data types in any modern language including C#. It is a sequential read-only collection of `Char` objects
- `string` is an alias for `String`. Therefore, `string` and `String` are equivalent.
- String objects are immutable. They cannot be changed after created.

```
String str1 = "Hello World!";
str1 = "Hello World!! from Tutorials Teacher";
```

Fresher Academy

- String Escape Sequences: khái niệm Escape để chỉ ra nghĩa thay thế cho nghĩa thông thường. Trong chuỗi khi ta compile 1 dấu backslash và tiếp theo bởi 1 kí tự thì sẽ gọi là escaped sequences

String Escape Sequences

- Escaping gives an alternative meaning to the "normal" meaning. Character combinations consisting of a backslash followed by a letter or digits called "escaped sequences".

Short Notation	UTF-16 character	Description
\'	\u0027	allow to enter a ' in a character literal e.g. '\''
\"	\u0022	allow to enter a " in a string literal e.g. "this is the double quote ("") character"
\\	\u005c	allow to enter a \ character in a character or string literal e.g. '\\\' or "this is the backslash (\\\\") character"
\n	\u000a	line-feed (next line)
\r	\u000d	carriage-return (move to the beginning of the line)
\t	\u0009	(horizontal-) tab
\v	\u000b	vertical-tab

Fresher Academy

- Khai báo và khởi tạo 1 chuỗi
 - Datatype + tên biến = gán giá trị
 - Thường hay dùng regular string literal để khai báo
 - Dùng vébatim string literal để khởi tạo(thêm @ trước chuỗi)

Declaration and Initialize String

```
// Initialize as an empty string.  
string message1 = System.String.Empty;  
string message2 = "";
```

```
//Initialize with a regular string literal.  
string oldPath = "c:\\Program Files\\Microsoft Visual Studio 8.0";
```

c:\\Program Files\\Microsoft Visual Studio 8.0
c:\\Program Files\\Microsoft Visual Studio 9.0

```
// Initialize with a verbatim string literal.  
string newPath = @"c:\\Program Files\\Microsoft Visual Studio 9.0";
```



Fresher Academy



- Common method cho string
 - Concat: Combine 2 string lại với nhau và trả về string mới
 - Contains: trả về giá trị kiểu int để biết rằng 1 chuỗi có nằm trong 1 chuỗi khác hay không
 - String.Format: convert giá trị của 1 object trở thành kiểu chuỗi dựa trên format mà chúng ta chỉ định(Hay dùng)
 - String.Substring: trả lại 1 chuỗi con từ 1 chuỗi to

Common Methods

<https://docs.microsoft.com/en-us/dotnet/api/system.string?view=netframework-4.7.1#Methods>

Name	Description
Concat	Combine two or more strings into a new string
Contains	Returns a value indicating whether a specified sub string occurs within this string
String.Format	Converts the value of objects to strings based on the formats specified and inserts them into another string
String.Substring	Retrieves a substring from a string.

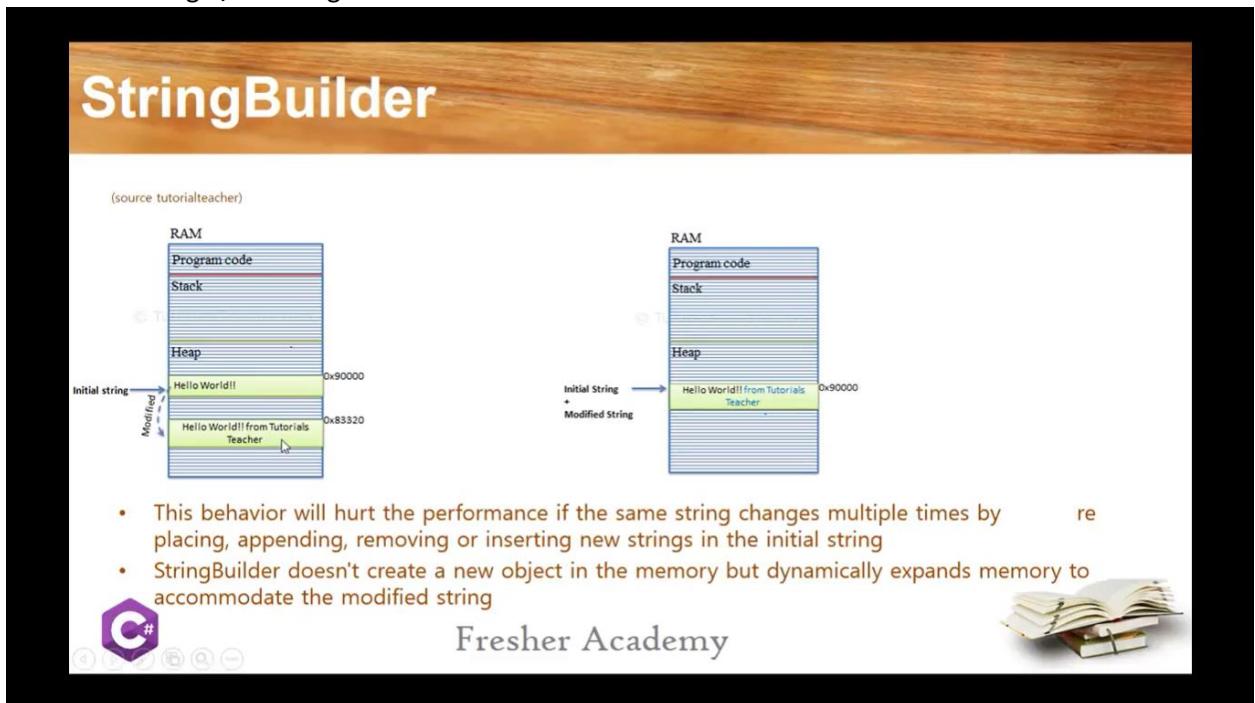


Fresher Academy



5 9 StringBuilder Video

- String là immutable nghĩa là khi gán giá trị vào string và chúng ta thay đổi giá trị đó thì sẽ tạo ra vùng nhớ mới trong heap cho giá trị thay đổi chứ nó sẽ không ghi đè lại vùng nhớ cũ
- Hành động nói trên có thể gây ảnh hưởng tới performance và để giải quyết vấn đề đó thì .NET Framework đã đưa ra StringBuilder
- StringBuilder: khi tạo ra object mới thì nó sẽ tự động resign lại memory ở trong heap cho giá trị mới và sẽ không tạo ra vùng nhớ mới cho chuỗi mới



- Cú pháp để khởi tạo StringBuilder:
 - `StringBuilder sb = new StringBuilder();`
//or
 - `StringBuilder sb = new StringBuilder("Hello World!!");`

Initialize

```
Example: StringBuilder  
  
StringBuilder sb = new StringBuilder();  
//or  
StringBuilder sb = new StringBuilder("Hello World!!");
```

```
Example: StringBuilder  
  
StringBuilder sb = new StringBuilder(50);  
//or  
StringBuilder sb = new StringBuilder("Hello World!!",50);
```



- **Stringbuilder Method**
 - `StringBuilder.Append(ValueToAppend)`: cho phép nối chuỗi vào trong chuỗi hiện tại, vào vị trí cuối cùng của chuỗi hiện
 - `StringBuilder.AppendFormat()`: thay đổi format của chuỗi hiện tại
 - `StringBuilder.Insert(index,ValueToAppend)`: insert chuỗi mới vào chuỗi hiện tại, chỉ định index vị trí muốn insert
 - `StringBuilder.Remove(int startIndex, int length)`: xoá chuỗi
 - `StringBuilder.Replace(oldValue,newValue)`: replace tới giá trị chuỗi mới và cũ.

PatternMatchingVideo

- ParternMatching: toán tử `is` để kiểm tra xem object của chúng ta có thuộc type nào hay không

Pattern Matching

- Often you need to work with the *object* type. C# 7.0 offers a features for pattern matching: the **is** operator. This operator can be used to check if an object is compatible with a specific type.

```
object[] data = { null, 42, new Person("Ronaldo")};

foreach (var item in data)
{
    IsPattern(item);

    if (o is null) Console.WriteLine("it's a const pattern");
    if (o is 42) Console.WriteLine("it's 42");
    if (o is int i) Console.WriteLine($"it's a type pattern with an int and the value {i}");
    if (o is Person p) Console.WriteLine($"it's a person: {p.FirstName}");
}
```



Fresher Academy



String Interpolation Video

Old ways of Concating

```
Customer customer = new Customer();
customer.FirstName = "Lionel";
customer.LastName = "Messi";
customer.Age = 40;

// Concat string by using +
Console.WriteLine("FullName is " + customer.FirstName + " " + customer.LastName + ". His age is " + customer.Age + " years old.");

// Contact string by using string.Format
Console.WriteLine(string.Format("\nFullName is {0} {1}. His age is {2} years old.", customer.FirstName, customer.LastName, customer.Age));
```

```
FullName is Lionel Messi. His age is 40 years old.  
FullName is Lionel Messi. His age is 40 years old.
```



Fresher Academy



- String Interpolation: giống như một template chứa biểu thức nội suy và khi interpolation trả lại chuỗi sẽ tự động replace những biểu thức này với giá trị mà biểu thức này biểu diễn

Old ways of Concatenating

```
Customer customer = new Customer();
customer.FirstName = "Lionel";
customer.LastName = "Messi";
customer.Age = 40;

// Concat string by using +
Console.WriteLine("FullName is " + customer.FirstName + " " + customer.LastName + ". His age is " + customer.Age + " years old.");

// Contact string by using string.Format
Console.WriteLine(string.Format("\nFullName is {0} {1}. His age is {2} years old.", customer.FirstName, customer.LastName, customer.Age));
```

FullName is Lionel Messi. His age is 40 years old.
FullName is Lionel Messi. His age is 40 years old.



Fresher Academy



3 0 OOP Introduction edited

- What is OOP: Object oriented Programming
Lập trình hướng đối tượng cho phép lập trình viên có thể viết object trong source code, giả lập obj hay đối tượng trong đời sống hằng ngày
- Object là bất cứ thứ gì có attribute hay thuộc tính và có phương thức(behavior)
- Ví dụ:

Object Example

- It has a name
- It has color
- It has weight/height
- It can eat
- It can walk
- It can makes a sound



Fresher Academy



- Class là blueprint hoặc template(khuôn mẫu) để từ đó chúng ta có thể tạo ra object

What is a Class?

- Class is a blueprint or template from which lots of individual objects can be created.

Fresher Academy

- Ví dụ về class

Class Example

Fresher Academy

3 1 ClassDesign

- Class, Object Recall: Class là một khuôn mẫu mà từ đó ta có thể tạo ra được object
- Còn một object có thể hiểu là khởi tạo, là một instance của class
- Designing a class: để thiết kế ra một class thì cần nghĩ về object mà sẽ được tạo ra từ class đó

- Một object gồm 2 thành phần bao gồm : attribute(thuộc tính) và behavior(phương thức)

Designing a class

- When you design a class, think about the objects that will be created from that class.
 - Things the object **knows** about itself
 - Things the object **does**

C# Fresher Academy

The diagram shows a class box for `ShoppingCart`. It has two sections: `cartContents` (labeled **knows**) and a list of methods (`addToCart()`, `removeFromCart()`, `checkOut()`) (labeled **does**).

- Ví dụ một giỏ hàng có những hành động thêm product vào giỏ hàng thì chúng ta tạo ra method `addToCart()`, `RemoveFromCart()`, `CheckOut`
- Instance variables là những thứ cho biết một object biết về bản thân nó hoặc gọi là properties hoặc attributes
- Những object instance variable thể hiện ra trạng thái của object đấy
- Object can do thì chúng ta gọi là method và tập hợp method này sẽ biểu thị ra hành vi hay còn gọi behavior của obj
- Ví dụ về class

Writing a class

1. Write the class

```
class Dog {  
    int size;  
    String breed;  
    String name;  
  
    void bark() {  
        System.out.println("Ruff! Ruff!");  
    }  
}
```

instance variables

a method

DOG
size
breed
name
bark()



Fresher Academy



Writing the class (2)

2. Write a test class

dot notation (.) gives access to an object's instance variables and methods

```
public class DogTestDrive {  
    public static void main(String [] args) {  
        Dog d = new Dog();  
        d.name = "Bruno";  
        d.bark();  
    }  
}
```

make a Dog object

set the name of the Dog

call its bark() method



Fresher Academy



3 2 Constructor

- Constructor được sử dụng để tạo ra object từ class, có thể coi constructor là member function đặc biệt vì :
 - Nó có cùng tên với tên class
 - Constructor được gọi bất cứ khi nào object được tạo ra
 - Constructor không có kiểu giá trị trả về

- Constructor là function nên cho phép truyền parameters vào function đó
- Constructor có thể overloaded
- Khi mà chúng ta không định nghĩa constructor trong 1 class thì compiler tự động tạo ra Constructor đặc biệt gọi là Default Constructor và Default Constructor sẽ không có parameter nào cả

3 3 Namespace

- Namespace được thiết kế để tránh class trùng tên với nhau và giảm thiểu conflict giữa các class. Khi mà một class được định nghĩa trong một namespace thì nó sẽ không bị conflict với class trùng tên
- Ví dụ về Namespace

```

1  using System;
2  namespace International
3  {
4      2 references
5      class Sale
6      {
7          1 reference
8          public void detail()
9          {
10             Console.WriteLine("This is international sale!!!");
11         }
12     }
13 }
14 namespace Domestic
15 {
16     2 references
17     class Sale
18     {
19         1 reference
20         public void detail()
21         {
22             Console.WriteLine("This is domestic sale!!!");
23         }
24     }
25 }
26 class Program
27 {
28     0 references
29     static void Main(string[] args)
30     {
31         International.Sale iSale = new International.Sale();
32         Domestic.Sale dSale = new Domestic.Sale();
33         iSale.detail();
34         dSale.detail();
35     }
}

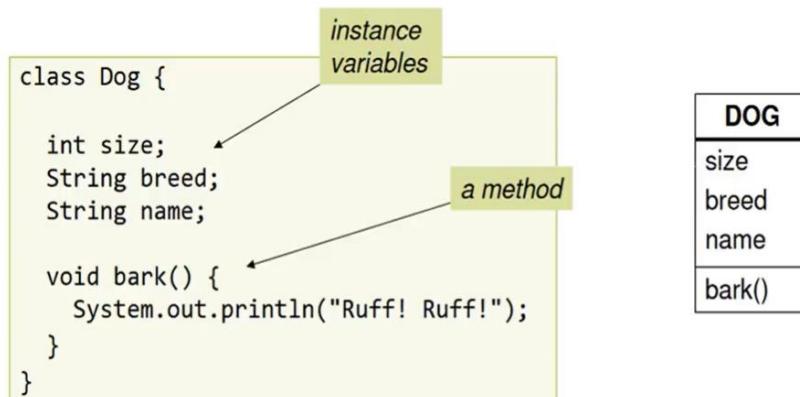
```

3 4 Class Lab Edited

Writing a class

1. Write the class

-



3.5 Properties And Methods

- Object có 2 thành phần chính:
 - State (Trạng thái), state được biểu thị thông instance variable hay còn gọi là properties hoặc attribute
 - Hành xử của object (Behavior) (Method)
- Những method trong 1 class có thể sử dụng attribute value và nó sẽ thay đổi object state đó
- Chính vì thay đổi được state đó nên những object có cùng 1 kiểu có thể hành xử khác nhau

State affects behavior and vice versa.

The diagram shows a C# class definition for a Dog:

```

class Dog {
    int size;
    String breed;
    String name;

    void bark() {
        if (size > 14)
            System.out.println("Ruff! Ruff!");
        else
            System.out.println("Yip! Yip!");
    }

    void getBigger() {
        size += 5;
    }
}

```

Annotations explain the relationship between state and behavior:

- A callout points to the bark method with the text: "State affects behavior. Dogs of different sizes behave differently."
- A callout points to the getBigger method with the text: "method changes state".
- A callout at the bottom right points to the code with the text: "fresner Academy".

To the right is a table labeled "DOG" with columns for size, breed, name, bark(), and getBigger().

State affects behavior and vice versa.

The diagram shows a C# test drive code for a Dog:

```

class DogTestDrive {
    public static void main (String[] args) {
        Dog one = new Dog();
        one.size = 7;
        Dog two = new Dog();
        two.size = 13;

        two.bark();
        two.getBigger();
        two.bark();

        one.bark();
    }
}

```

Annotations show the state of two objects:

- An arrow labeled "one" points to a box labeled "Dog object 1" with fields: name:null, size: 7, breed:null.
- An arrow labeled "two" points to a box labeled "Dog object 2" with fields: name:null, size:18, breed:null.

Output text on the right shows the results of the bark and getBigger methods:

```

Yip! Yip!
Ruff! Ruff!
Yip! Yip!
%>

```

A callout at the bottom right points to the code with the text: "fresner Academy".

- Giữa method và properties có quan hệ 2 chiều, thì method có thể thay đổi properties của 1 object và ngược lại properties có thể ảnh hưởng đến method. Chính vì thế object có cùng kiểu nhưng có thể hành xử khác nhau

3 6 Encapsulation

- Tính đóng gói: được sử dụng để che giấu thông tin trong 1 object, ví dụ có 1 attribute và không muốn visible ra bên ngoài thì chúng ta có thể bundle nó vào trong method và method đó sẽ đọc và ghi tới attribute đó và thế giới bên ngoài sẽ chỉ làm việc với method này thôi và không làm việc với attribute
- Encapsulation/ Information Hiding

Encapsulation / Information Hiding (2)

- How to prevent creating a supernatural dog?
 - Write `set` method for attributes
 - Hide the attributes to force other code to use `set` method instead of accessing them directly

```
class Dog {
    private int size;

    public void setSize(int s) {
        if (s > 0) size = s;
    }

    public int getSize() {
        return size;
    }
    ...
}
```





- Set thuộc tính method là private
- Ưu điểm của Advantages
 - Tất cả độ phức tạp của code sẽ được ẩn giấu bên trong object và client chỉ việc sử dụng mà không cần biết đến bên trong
 - Gây ra ít lỗi

3.7 Access Modifiers

- Access Modifiers là keywords set ra mức độ truy cập đến class, method, và attribute. accessModifiers được đưa ra để thực hiện tính đóng gói trong oop
- C# cung cấp 4 loại accessmodifiers chính:
 - Public: mức độ truy cập ở bất cứ đâu và bởi bất cứ ai, độ bảo mật thấp nhất
 - Protected: khi một class,method,attribute được set là protected thì có nghĩa là class,method,attribute này chỉ được truy cập bởi chính class đó hoặc class con kế thừa từ class cha
 - Internal: khi sử dụng internal thì quyền truy cập chỉ trong nội assembly
 - Private: chỉ truy cập nội bộ trong object đó và bên ngoài không thể truy cập vào
- Khi không chỉ định ra accessmodifiers cho class thì mặc định là internal
- Default access modifiers cho attribute,method, constructor của class là private

Example

- How to prevent creating a supernatural dog?
 - Write `set` method for attributes
 - Hide the attributes to force other code to use `set` method instead of accessing them directly

```
class Dog {  
    private int size;  
  
    public void setSize(int s) {  
        if (s > 0) size = s;  
    }  
  
    public int getSize() {  
        return size;  
    }  
    ...  
}
```



Fresher Academy



3.9 Inheritance

- Tính kế thừa: sẽ cho phép bạn định nghĩa ra lớp con có thể sử dụng lại hoặc mở rộng sửa đổi behavior lớp cha. Tính kế thừa này cũng giống như trong thực tế ngoài đời song: ví dụ con cái kế thừa đặc tính của cha mẹ như hình dáng, màu mắt, màu tóc
- Những class mà member của nó được kế thừa thì chúng ta gọi là lớp cha hay còn gọi là base class
- Còn những lớp con kế thừa từ lớp cha thì chúng ta gọi là lớp con hay derived
- C# nói riêng và .NET nói chung chỉ hỗ trợ single inheritance có nghĩa là một lớp con chỉ có thể kế thừa từ một lớp cha duy nhất, nhưng một lớp cha thì có thể có nhiều lớp con
- Ví dụ:

Design Animal simulation



C# Fresher Academy

1. Tìm ra đặc tính chung của object
 - Picture
 - Food
 - Hunger: biểu thị mức độ đói của con vật đó
 - Boundaries: vùng không gian cho phép con vật đó đi lại
 - Location: tọa độ

Design Animal simulation (1)

1. Look for objects that have common attributes:
 - **picture** – the file name representing the JPEG of this animal
 - **food** – the type of food this animal eats. Right now, there can be only two values: *meat* or *grass*.
 - **hunger** – representing the hunger level of the animal. It changes depending on when (and how much) the animal eats.
 - **boundaries** – values representing the height and width of the 'space' (for example, 640 x 480) that the animals will roam around in.
 - **location** – the X and Y coordinates for where the animal is in the space.

picture
food
hunger
boundaries
location

C# Fresher Academy

2. Tìm ra behavior chung:
 - makeNoise(): tạo ra âm thanh

- eat(): việc ăn của từng loại con vật
- sleep(): ngủ
- roam: kiểu đi lang thang của từng con

Design Animal simulation (2)

2. Look for objects that have common behaviors:

- ***makeNoise()*** – behavior for when the animal is supposed to make noise.
- ***eat()*** – behavior for when the animal encounters its preferred food source, *meat* or *grass*.
- ***sleep()*** – behavior for when the animal is considered asleep.
- ***roam()*** – behavior for when the animal is not eating or sleeping (probably just wandering around waiting to bump into a food source or a boundary).

Fresher Academy




3. Design class

Design Animal simulation (3)

3. Design a class that represents the common state and behavior:
These objects are all animals, so we'll make a common superclass called **Animal**

Animal
picture food hunger boundaries location
makeNoise() eat() sleep() roam()

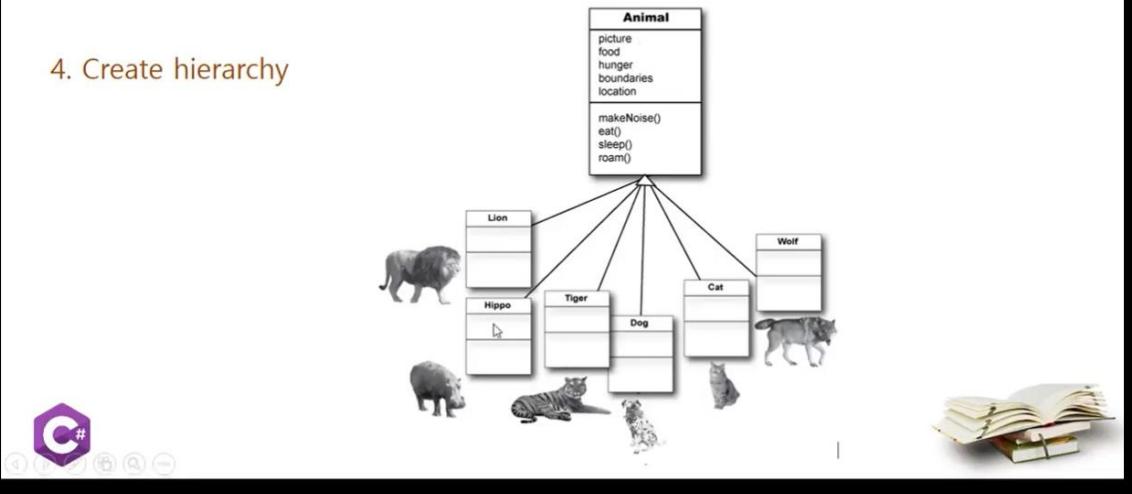
Fresher Academy




4. Tạo hierarchy

Design Animal simulation (4)

4. Create hierarchy

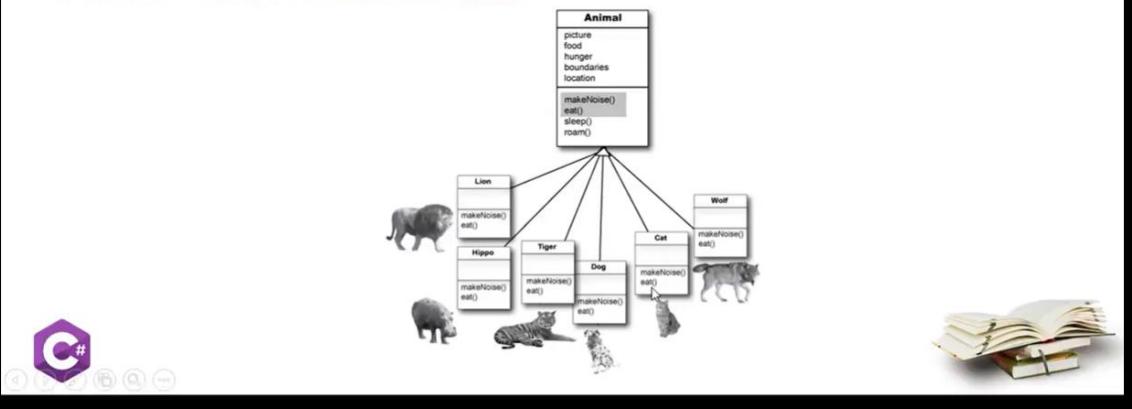


5. Những behavior nào nên overrides

- Overrides Phương thức makeNoise, eat

Design Animal simulation (5)

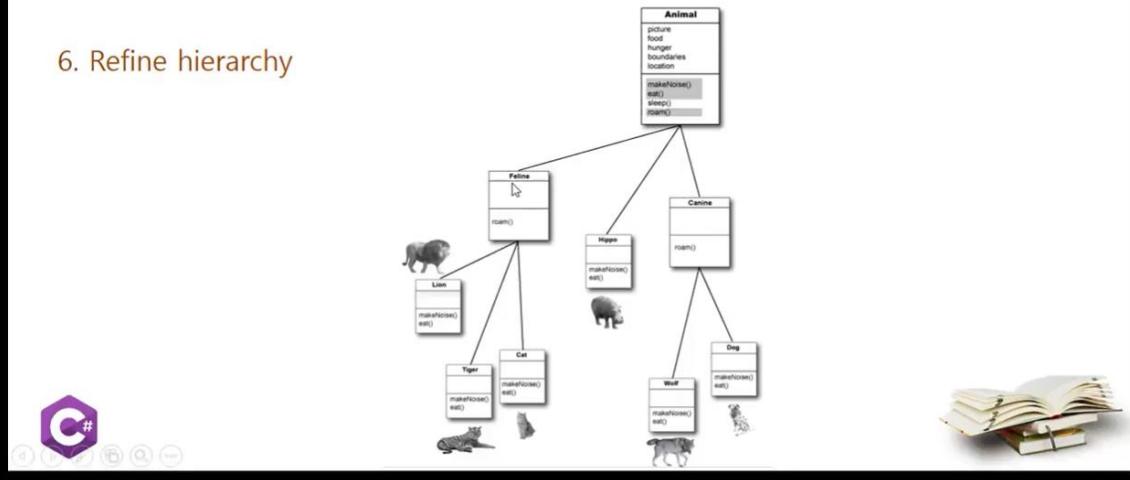
5. Which behaviors should overrides?



6. Refine hierarchy

Design Animal simulation (6)

6. Refine hierarchy



- Nên overrides lại phương thức `roam()`
- Lợi ích của tính kế thừa
 - Nhờ có kế thừa mà chúng ta có thể minimize được code bị trùng lặp, tránh dư thừa
 - Data hiding: có những dữ liệu mà chúng ta có thể private ở lớp cha mà lớp con không thể thay đổi được
 - Overriding: lớp cha define ra phương thức và thuộc tính nhưng việc implement một cách tường minh thì chúng ta sẽ để ở lớp con

3 12 Abstraction

- Abstraction: tính trừu tượng hóa sẽ tập trung vào quá trình che giấu đi những chi tiết không cần thiết và sẽ thể hiện ra bên ngoài đặc tính cơ bản của object cụ thể
- Ví dụ class car được tạo ra bởi rất nhiều thành phần như động cơ, hộp số, vô lăng.v.v... tuy nhiên khi nói đến oto chỉ nói đến thành phần cơ bản nhất như nó có động cơ, hộp số thay vì đi sâu vào bên trong và đưa ra vấn đề làm sao để oto chạy và khởi động động cơ. Thị đây chính là trừu tượng hóa
- Về mặt ngữ nghĩa thì trừu tượng có nghĩa là liên quan đến ý tưởng, concept hơn là liên quan đến instance cụ thể
- Trong lập trình chúng ta có thể áp dụng tính trừu tượng bằng cách tạo ra class sẽ không liên quan gì đến instance cụ thể nào

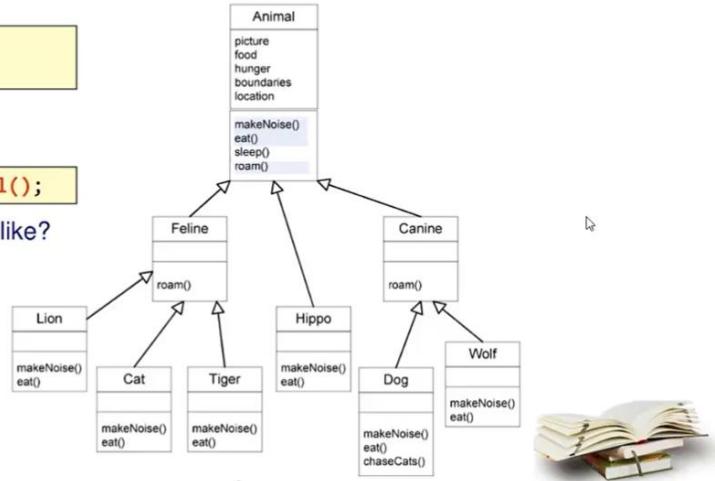
Example

```
Dog d = new Dog();  
Cat c = new Cat();
```

Fine. But...

```
Animal anim = new Animal();
```

What does an Animal look like?



↗



Animal ở đây là abstraction class

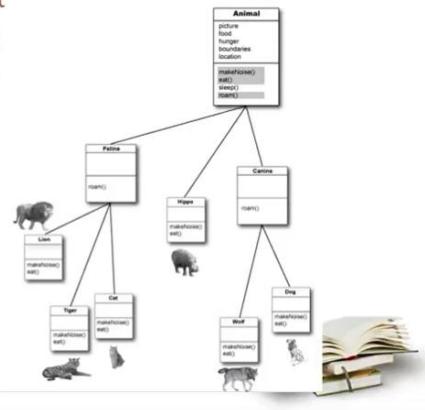
- Abstraction vs Encapsulation
 - Cả 2 đều nhằm mục đích che giấu thông tin tới thế giới bên ngoài
- Khác biệt giữa Abstraction vs Encapsulation
 - Abstraction giải quyết việc che giấu thông tin ở mức design. Encapsulation giải quyết việc che giấu thông tin ở mức triển khai(implementation level)
 - ở mức thiết kế abstraction có thể tạo ra class là abstraction bằng cách sử dụng từ khoá abstract hoặc sử dụng interface. Còn Encapsulation sẽ được implement bằng cách sử dụng access modifiers(Private, protected)

3.14 Polymorphism

- Polymorphism: tính đa hình có nghĩa là 1 object có thể hành xử dưới nhiều dạng thức khác nhau hoặc chúng ta có thể nói là với cùng 1 method và 1 properties thì nó có thể thực thi khác nhau phụ thuộc vào loại object ở thời điểm runtime thực thi method và properties đó
- Ví dụ: cùng 1 người thì người đó có thể hành xử như một người con ở trong gia đình, tuy nhiên cùng thời điểm thì người đó hành xử như một nhân viên tại văn phòng

Polymorphism

- It means one object behaving as multiple forms. With polymorphism the same method or property can perform different actions depending on the run-time type of the instance that invokes it
 - Example: Person behaves as a SON in house, at person behaves like an EMPLOYEE in the office.



Fresher Academy

```
//leo.MakeNoise();
//leo.Eat();
//leo.Roam();
Animal animal1;
Animal animal2;
Animal animal3;

animal1 = new Lion();
animal2 = new Cat();
animal3 = new Wolf();

animal1.Eat();
animal1.MakeNoise();
animal1.Roam();

animal2.Eat();
animal2.MakeNoise();
animal2.Roam();

animal3.Eat();
```

I'm a lion and I eat
I'm a lion Gruuu...
We go in a pack
I'm a cat and I eat
I'm a cat Meow meow...
We go in a pack
I'm a wolf and I eat
I'm a wolf uhhhhhh uhhhhhhh...
We go alone

- ⇒ Cùng gọi đến 1 phương thức nhưng kết quả hiện ra lại tùy thuộc vào instance lúc tạo thì đây chính là tính đa hình

3 15 Interface

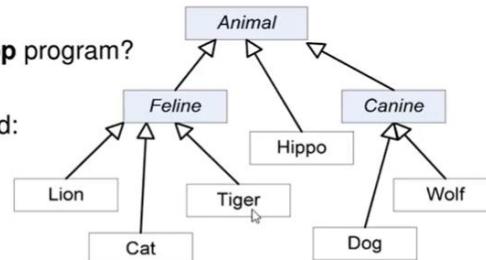
A new Challenge

designed for an animal simulation program
reusable in educational software in zoology

What about a **PetShop** program?

Pet behaviors required:

- `beFriendly()`
- `play()`



Fresher Academy



- Ví dụ yêu cầu chương trình PetShop phải có 2 behavior
 - `beFriendly()`
 - `Play()`

Option 1

Put `beFriendly()` and `plays()` code up here for inheritance

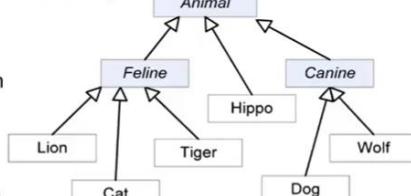
Put all the pet method code up

Pros:

- Pet polymorphism
- Code inherited

Cons:

- Hippos as pets?
- Lions and Wolves, too?
- We still have to override pet methods in Cat and Dog



Fresher Academy



- Xử lý: tiếp cận từ trên xuống thêm 2 phương thức ở class animal
 - Ưu điểm của cách này bao gồm: tất cả class đều đc kế thừa và chương trình PetShop hoàn thành được tính đa hình của nó
 - Nhược điểm: class Hippo, lion, wolf cũng là vật nuôi thì nghe có vẻ không hợp lý

Option 2

Put the pet methods ONLY in the classes where they belongs

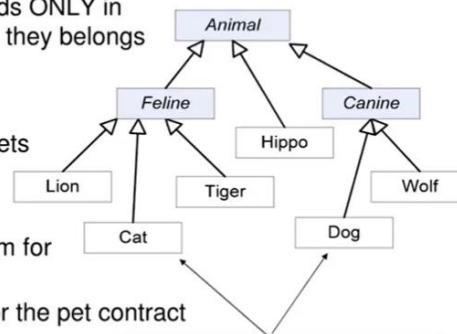
Pros:

- No Hippos as pets

Cons:

- no polymorphism for pet methods
- no guarantee for the pet contract

Put the pet methods ONLY in the classes that can be pets



Fresher Academy

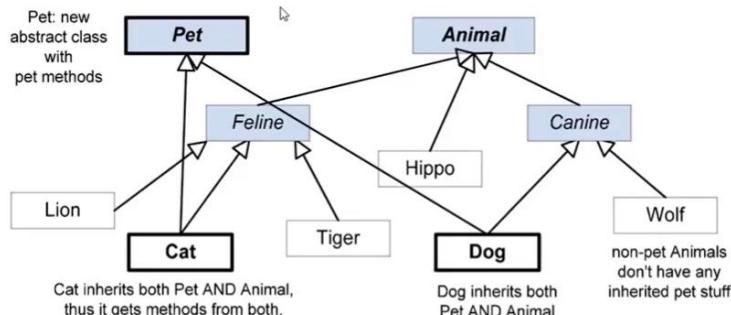


- Cách tiếp cận thứ 2: đi từ dưới lên, có thể con chó hoặc con mèo là vật nuôi thì chúng ta có thể đẩy method đó vào cat và dog
 - Ưu điểm là : Hippo,lion,wolf không còn là vật nuôi
 - Nhược điểm: tuy nhiên lại làm mất tính đa hình đã thiết kế trước đó

What we need

A way a pet behavior in **just** the pet class

Pet: new abstract class with pet methods

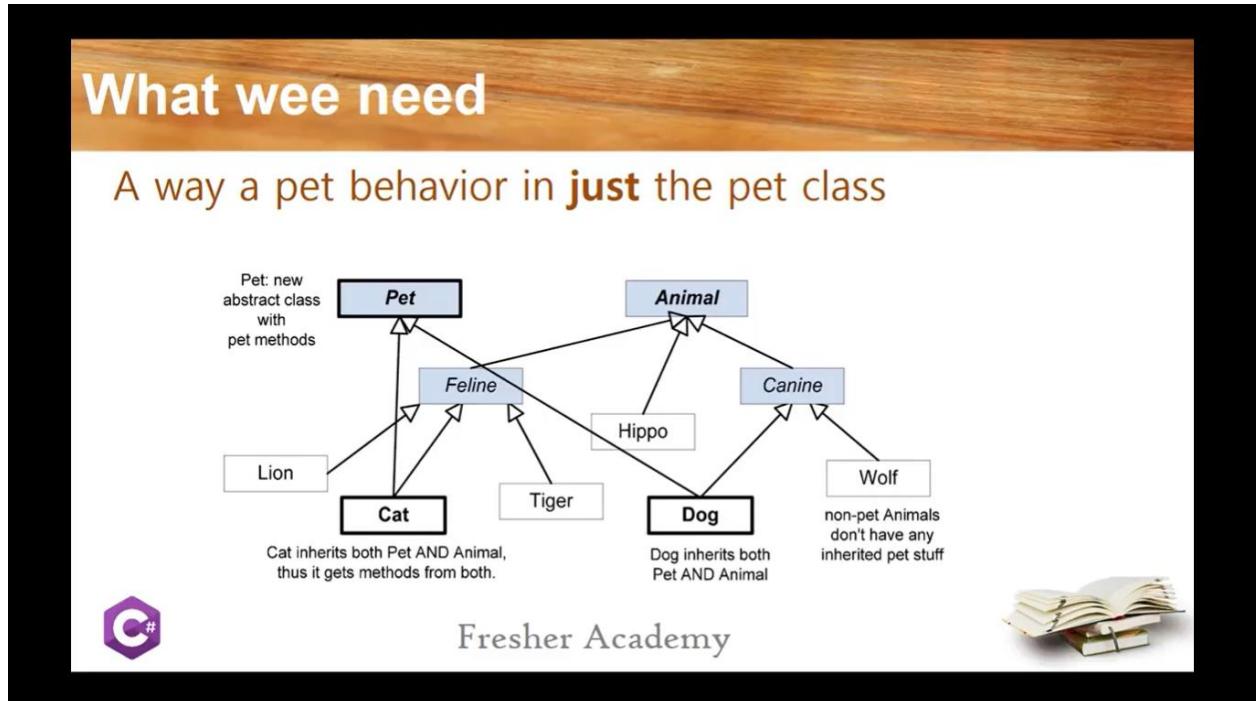


Fresher Academy



- ⇒ Những pet behavior chỉ nằm trong class pet mà thôi và những class nào vừa là pet thì sẽ phải kế thừa cả pet và animal, ví dụ cat ở đây kế thừa từ pet và kế thừa từ feline và dog cũng tương tự. Thì cách kế thừa này sẽ đảm bảo tính toàn vẹn(polimorphism) và tính kế thừa, tuy nhiên có 1 vấn đề 1 class chỉ đc kế thừa từ 1 class cha và giờ phải làm sao

- ⇒ Thì bây giờ khái niệm interface ra đời để vượt qua vấn đề trên. Interface sẽ định nghĩa ra các properties, method, event mà chúng ta gọi là member của interface và các interface này sẽ chỉ chứa các khai báo về các member này mà thôi. Còn việc implement chi tiết các member này sẽ do class kế thừa định nghĩa ra



- Quay lại vấn đề: không phải tạo ra class pet nữa mà tạo ra interface có tên là pet và lúc đó cat sẽ kế thừa từ pet và cả class Feline

13 OverloadVideo

- Polymorphism có nghĩa là object có thể hành xử dưới nhiều dạng thức khác nhau, trong C# có 2 loại Polymorphism
- Static Polymorphism: đa hình tĩnh thì việc quyết định gọi method nào của object tức là hành xử của object đấy sẽ được quyết định ở thời điểm compile. Chính vì vậy static polymorphism được gọi là compile polymorphism, để thực thi đa hình tĩnh này chúng ta cần phương pháp overloading(nạp chồng). Trong overloading có 2 kiểu
 - Method overloading:
 - Operator overloading:
- Dynamic Polymorphism: đa hình động quyết định xem gọi method vào, hành xử nào của 1 object được quyết định ở thời điểm runtime. Để thực thi Dynamic Polymorphism thì chúng ta sử dụng :
 - Method overriding
- Method overloading cho phép chúng ta tạo ra nhiều method trong cùng một class có cùng tên nhưng khác nhau về parameters và type truyền vào. Có thể thực thi method overloading bằng 1 trong 3 cách sau
 - Thay đổi số lượng parameter cho các method ở trong class đó
 - Thay đổi order thứ tự của parameter đó
 - Sử dụng loại datatype khác nhau cho parameter

- Ví dụ:

```
2 references
public class AddingWrapper
{
    1 reference
    public int add(int a, int b) //Method 1
    {
        return a + b;
    }

    0 references
    public int add(int a, int b, int c) //Method 2
    {
        return a + b + c;
    }

    0 references
    public float add(float a, float b, float c, float d) //Method 3
    {
        return a + b + c + d;
    }
}
```

C# Fresher Academy

13 3 Constructor Overloading Video

- Constructor Overloading là một công nghệ cho phép chúng ta có thể tạo ra nhiều constructor chỉ khác nhau ở tập hợp parameter truyền vào cho constructor. Với việc sử dụng constructor overloading thì cho phép chúng ta sử dụng class theo nhiều khía cạnh khác nhau

```
public class Rectangle
{
    2 references
    public int Height { get; set; }

    2 references
    public int Width { get; set; }

    0 references
    public Rectangle(int height, int width)
    {
        Height = height;
        Width = width;
        Console.WriteLine("Rectangle constructor is called");
    }

    1 reference
    public Rectangle(int size)
    {
        Height = Width = size;
        Console.WriteLine("Square constructor is called");
    }
}
```

- Tạo class rectangle thì trong đó có 2 properties là chiều cao và rộng, sau đó thì có một constructor truyền vào 2 biến chiều cao và chiều rộng để mình set giá trị cho chiều cao và rộng của rectangle này
- Tiếp theo có constructor thứ 2: làm nhiệm vụ tạo ra hình vuông(trường hợp đặc biệt), chiều cao rộng bằng nhau, thì với hình vuông lúc này sẽ chỉ cần truyền vào thông số là size và sau đó set chiều rộng và chiều cao cùng bằng giá trị là size
- ⇒ Tạo ra constructor overloading, tạo ra 2 constructor chỉ khác nhau ở số lượng parameter truyền vào và với việc tạo ra 2 constructor thì mình có thể sử dụng class như một hình chữ nhật hoặc sử dụng class này như hình vuông
- Constructor calling: trong thực tế tạo ra class có thể có rất nhiều properties trong class đó thì constructor có thể trở nên rất phức tạp, rất lớn với nhiều parameter và khi chúng ta sử dụng constructor overloading thì các constructor này sẽ lặp đi lặp lại khiến chúng ta chán nản khi code. Thì để tránh điều này thì C# cho phép chúng ta tạo ra một constructor bằng việc gọi một constructor sẵn có
 - Cú pháp:

public Constructor(parameters1) : this(parameters2) {}

Trong đó public access modifiers + constructor(truyền vào tham số): từ khoá this(truyền vào danh sách tham số). Lưu ý parameter 2 phải là tập con của parameter 1

```
public class Rectangle
{
    2 references
    public int Height { get; set; }
    2 references
    public int Width { get; set; }

    0 references
    public Rectangle(int height, int width)
    {
        Height = height;
        Width = width;
        Console.WriteLine("Rectangle constructor is called");
    }

    1 reference
    public Rectangle(int size)
    {
        Height = Width = size;
        Console.WriteLine("Square constructor is called");
    }
}
```

```
public class Rectangle
{
    1 reference
    public int Height { get; set; }
    1 reference
    public int Width { get; set; }

    1 reference
    public Rectangle(int height, int width)
    {
        Height = height;
        Width = width;
        Console.WriteLine("Rectangle constructor is called");
    }

    1 reference
    public Rectangle(int size) : this (size, size)
    {
        //Height = Width = size;
        Console.WriteLine("Square constructor is called");
    }
}
```

13 5 Method Overriding Video

- Method overriding được sử dụng để tạo ra dynamic polymorphism tức là 1 object có thể gọi đến một method của behavior của nó tại thời điểm runtime, để thực hiện việc đó method overriding cho phép chúng ta có thể tạo ra method ở lớp con có cùng tên, cùng parameter, cùng giá trị trả về giống như lớp cha(base class)
 - Method overriding chỉ được thực thi ở lớp con, chúng ta không thể overriding một method trong cùng 1 class
 - Một method ở lớp cha muốn cho lớp con có thể overriding thì method đó cần phải dùng 1 trong 2 từ khóa là virtual hoặc abstract
- Ví dụ:

Sample

```
public class Animal
{
    1 reference
    public virtual void eat()
    {
        Console.WriteLine("Eating...");
    }
}
0 references
public class Lion : Animal
{
    1 reference
    public override void eat()
    {
        Console.WriteLine("Eating meat...");
    }
}
```



Fresher Academy



13 7 Overload vs OverrideVideo

- So sánh Overload và Override
- Giống nhau:
 - Cùng thực thi tính đa hình trong lập trình hướng đối tượng
- Khác nhau:
 - ⇒ Overload:
 - Cho phép chúng ta tạo ra nhiều method trong cùng một class có cùng tên nhưng khác tham số truyền vào và kiểu dữ liệu truyền vào
 - ⇒ Override:
 - Cho phép chúng ta tạo ra method ở trong lớp con có cùng tên, cùng tham số và cùng kiểu dữ liệu trả về giống như lớp cha
 - Khi override một method ở lớp con thì method ở class chả phải kèm theo từ khoá virtual hoặc abstract
- Virtual Method
 - Virtual method có thể implement code trong virtual method và virtual method có thể nằm trong abstract class
 - Không bắt buộc lớp con phải override virtual method đó
- Abstract method
 - Không có implementation
 - Bắt buộc lớp con phải override bởi vì không có implementation

13 8 StaticMethodVideo

- Static method hay còn gọi là class method

Class Methods or Static Methods

```
double x = Math.Round(100.5);
int y = Math.Abs(-10);
Console.WriteLine("x: {0}", x);
Console.WriteLine("y: {0}", y);
```

```
x: 100
```

```
y: 10
```

- Math functions were written as **class methods** because they don't need to know about a specific Math object. This also called **static methods**.



Fresher Academy



- Cho ví dụ sau: có biến x kiểu double và sau đó gán giá trị của x = Math(class dựng sẵn của C#) và trong class Math này gọi đến method Round sẽ làm nhiệm vụ làm tròn con số(truyền vào là 100.5).
- Tiếp theo khai báo biến y kiểu int và cũng gọi method là abs(tính trị tuyệt đối trong class math) và truyền giá trị -10
⇒ Khi in giá trị x,y ra màn hình thì sẽ trả lại giá trị x = 100; y = 10
- Static method có thể hiểu là method có thể chạy mà không cần quan tâm tới instance

Instance Methods vs Class Methods

Instance (regular) methods

```
class Cow {
    String name;
    public String greeting() {
        return ("Hi, I am " + name);
    }
}
```

Class (static) methods

```
class Math {
    public static int abs(int a) {
        if (a > 0) return a;
        return -a;
    }
    ...
}
```

Regular method

Instance variable `name` affect variable to behavior of `greeting()`

MUST be called by an instance
`s = cow1.greeting()`

Static method

- `abs()` has nothing to do with Math variable

Be called by using class name
`int a = Math.abs(-10)`



Fresher Academy



- Regular method

- Một instance variable sẽ affect behavior của một method. Như trong ví dụ trên nếu thay đổi tên thì sẽ thay đổi nội dung greeting
- Regular method sẽ cần gọi nó từ instance
- Static method
 - Không thể access tới variable của class, nó sẽ chỉ quan tâm tới parameter truyền vào cho nó mà thôi
 - Có thể sử dụng luôn class và gọi đến method đó

Static Method can't use instance variable or instance method

```

0 references
public class CustomMath
{
    1 reference
    public Boolean hasPermission { get; set; }
    0 references
    public static int abs(int input)
    {
        if (hasPermission)
            return input > 0 ? input : (-input);
        else
            throw new Exception("You don't have permission to do this");
    }
}

if (hasPermission)
    return
else
    throw n
  
```

An object reference is required for the non-static field, method, or property 'CustomMath.hasPermission'

Fresher Academy

⇒ Một static method không thể sử dụng instance variable hoặc instance method

13 9 Static Variable Video

- Static variable còn được gọi là class variable, static variable phụ thuộc vào class chứ không phụ thuộc vào object cụ thể nào
- Sử dụng static variable như một bản copy được sử dụng giữa instance của class
- Ví dụ:

Sample

```
public class Duck
{
    0 references
    public String Name { get; set; }
    public static int Count = 0;

    3 references
    public Duck()
    {
        Count++;
    }
}
```

```
Console.WriteLine("Before any duck object is created, count is:{0}", Duck.Count);
var d1 = new Duck();
var d2 = new Duck();
var d3 = new Duck();
Console.WriteLine("After 3 duck objects are created, count is {0}", Duck.Count);
```

```
file:///C:/Users/CuongNguyen/Documents/Visual Studio 2015/Projects/C
Before any duck object is created, count is:0
After 3 duck objects are created, count is 3
```



Fresher Academy



13 10 StaticClassVideo

- Static class là một class mà chúng ta không thể tạo object từ nó(không thể instantiated). Một static class chỉ chứa một static member(chỉ chứa static variable hoặc static method). Static class có 2 đặc điểm:
 - Static class là sealed tức là không thể tạo ra class kế thừa từ static class
 - Static class không chứa instance constructor

Static Class

- A static class is a class that cannot be instantiated. It only contains static members
 - It is sealed.
 - Cannot contains instance constructor.
- A static class can be used as a convenient container for sets of methods that just operate on input parameters and do not have to get or set any internal instance fields

```
...public static class Math
{
    ...public const double E = 2.7182818284590451;
    ...public const double PI = 3.1415926535897931;

    ...public static decimal Abs(decimal value);
    ...public static double Abs(double value);
    ...public static float Abs(float value);
    ...public static int Abs(int value);
```



Fresher Academy



13 12 SealedVideo

- Sealed on a class: có thể áp dụng được ở mức class, method, properties, khi mà sử dụng từ khoá sealed cho một class thì chúng ta muốn ngăn class khác có thể kế thừa class sealed

Sealed on a Class

- When apply to a class, sealed prevent other classes from inheriting from it

```
class A {}  
sealed class B : A {}
```



Fresher Academy



- Sử dụng sealed cho method hoặc properties đã override virtual method hoặc property ở lớp base class
- Sử dụng sealed trên method và property cho phép class khác kế thừa class của bạn, tuy nhiên ngăn class kế thừa override virtual method

Sealed on Method and Property

- It can apply to a method or property that overrides a virtual method or property in a base class.
- This enables you to allow classes to derive from your class and prevent them from overriding specific virtual methods or properties

```
class X  
{  
    protected virtual void F() { Console.WriteLine("X.F"); }  
    protected virtual void F2() { Console.WriteLine("X.F2"); }  
}  
class Y : X  
{  
    sealed protected override void F() { Console.WriteLine("Y.F"); }  
    protected override void F2() { Console.WriteLine("Y.F2"); }  
}  
class Z : Y  
{  
    // Attempting to override F causes compiler error CS0239.  
    // protected override void F() { Console.WriteLine("Z.F"); }  
  
    // Overriding F2 is allowed.  
    protected override void F2() { Console.WriteLine("Z.F2"); }  
}
```



Fresher Academy



13 13 AutomaticPropertiesVideo

- Có ví dụ về autoProperties

The screenshot shows three snippets of C# code side-by-side:

```
public class Person
{
    private string _firstName;
    public string FirstName
    {
        get
        {
            return _firstName;
        }
        set
        {
            _firstName = value;
        }
    }
    private string _lastName;
    public string LastName
    {
        get
        {
            return _lastName;
        }
        set
        {
            _lastName = value;
        }
    }
    private readonly string ID;
```

```
public class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    private string ID { get; }
```

```
public class Person
{
    public string FirstName { get; set; } = "Tien Dung";
    public string LastName { get; set; } = "Bui";
    private string ID { get; }
```

Fresher Academy

⇒ Get là trả lại giá trị, set là gán giá trị

10 1 ExceptionVideo

- Exception: chính là vấn đề nảy sinh khi chúng ta thực thi chương trình, một exception trong C# chính là phản hồi về tình huống ngoại lệ xảy ra khi chương trình đang chạy
- C# cung cấp cho chúng ta cách xử lý exception bằng cách sử dụng 4 keywords là : try, catch, finally và throw
- Ví dụ về exception:

Sample

- The software within the car was able to detect a stopped vehicle. The software was designed to recognize this as a known problem, or as **an exception to normal behavior**



- When you go to an ATM to withdraw money, you expect to actually get money. What if you don't have any money in your account? That would be a good example of exception.



Fresher Academy



Chương trình cho xe tự lái, thì chương trình cần phải phát hiện ra xe khác đang chạy đằng trước bỗng dung dừng lại, và phải thiết kế phần mềm coi như đây là vấn đề ngoại lệ

10 2 ExceptionvsErrorVideo

So sánh exception vs Error

- Khi mà viết code hoặc thực thi chương trình trong C# thì nó thường có những loại lỗi sau xảy ra
 - Syntactical Error: lỗi cú pháp, ví dụ quên dấu ; và dùng ide để check lỗi
 - Compilation Error: lỗi trong quá trình compile
 - Runtime Error: chỉ xảy ra khi thực thi chương trình, ví dụ cố gắng đọc file mà không được cấp quyền
 - Lỗi runtime error chính là một exception bởi vì nó sẽ terminate chương trình đang chạy và để tránh termination chương trình đang chạy thì chúng ta cần handle exception đó.

10 3 TryCatchVideo

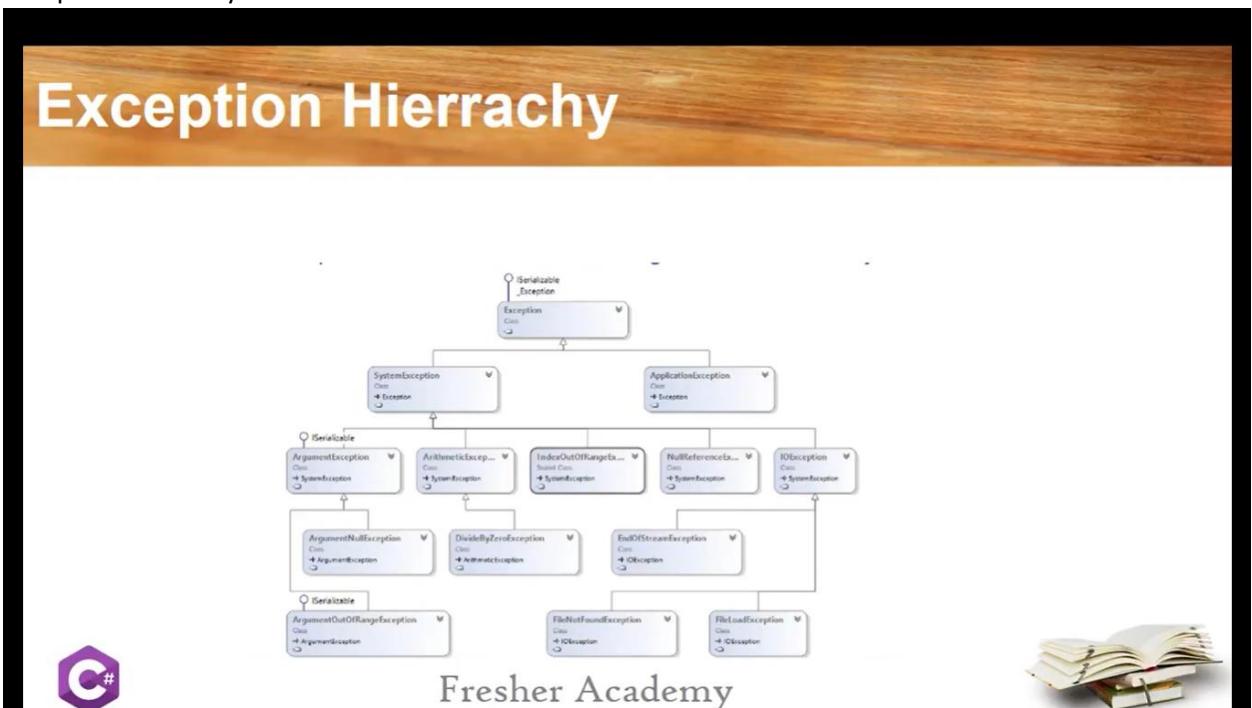
- Nhắc lại lý thuyết: exception chính là runtime error và nó sẽ terminate chương trình đang chạy, để tránh chương trình bị terminate thì chúng ta cần handle exception đó với keywords: try-catch, finally, throw
- Cấu trúc của try-catch: gồm 2 phần
 - Try: là block của code mà nó có thể gây ra exception và đi theo sau try-block thì có một hoặc nhiều catch
 - Catch: là nơi mà bắt exception và xử lý exception
 - Cú pháp:

```

try {
    // statements causing exception
} catch( ExceptionName e1 ) {
    // error handling code
} catch( ExceptionName e2 ) {
    // error handling code
} catch( ExceptionName eN ) {
    // error handling code
}

```

- Exception classes
 - Exception trong C# được biểu diễn bởi class và các class này sẽ kế thừa trực tiếp hoặc gián tiếp từ class system.exception
 - 2 class kế thừa từ system.exception đó là system.applicationException và system.SystemException
- Exception hierarchy



- Exception Properties
Exception có những thuộc tính sau:
 - Message: cung cấp chi tiết nguyên nhân gây ra lỗi
 - stackTrace:
 - InnerException: khi catch một exception thì có nhiều khả năng chúng ta throw lại exception mới, thì khi mà chúng ta Re-write một exception mới thì chúng ta có thể pass original exception vào trong InnerException
 - HelpLink: đưa ra một số link C# dụng sẵn

- Data: thuộc tính cho phép ta lock thêm một số thông tin hữu ích, hiển thị dữ liệu dưới dạng key value
- TargetSite: chỉ ra tên method gây ra exception

10 6 FinallyVideo

- Finally: dùng để cleanup object mà nó holding external resources, việc cleanup object này sẽ diễn ra ngay lập tức ngay cả khi exception được throw ra
- Ví dụ: chương trình thao tác với database mà chúng ta phải tạo ra connect tới db thì object lưu trữ connection tới db đó thì chúng ta gọi đó là object holding external resources tức là nguồn tài nguyên bên ngoài, số lượng connect đến db có hữu hạn chính vì vậy khi sử dụng xong thì phải đóng lại connection đó. Trong trường hợp chương trình gặp exception đó, trước khi mà chúng ta đóng connection thì chúng ta phải đợi barrage collection sẽ cleanup object thì có thể diễn ra chậm hơn và gây ra trường hợp giới hạn connect db chính vì vậy phải sử dụng finally. Bởi vì finally sẽ cleanup object ngay lập tức
- Ví dụ code:

```

static void CodeWithCleanup()
{
    System.IO.FileStream file = null;
    System.IO.FileInfo fileInfo = null;

    try
    {
        fileInfo = new System.IO.FileInfo("C:\\file.txt");

        file = fileInfo.OpenWrite();
        file.WriteByte(0xF);
    }
    catch(System.UnauthorizedAccessException e)
    {
        System.Console.WriteLine(e.Message);
    }
    finally
    {
        if (file != null)
        {
            file.Close();
        }
    }
}

```

Fresher Academy

10 7 ThrowVideo

- Tự hỏi rằng làm thế nào để ném ra một exception
- Trả lời: sử dụng câu lệnh throw
- Trường hợp tự throw ra exception
 - Khi method không thể hoàn thành chức năng mà chúng ta định nghĩa cho nó, ví dụ: một parameter cho method truyền vào giá trị invalid

Use cases

- The method cannot complete its defined functionality
 - For example, if a parameter to a method has an invalid value

```
static void CopyObject(SampleClass original)
{
    if (original == null)
    {
        throw new System.ArgumentException("Parameter cannot be null", "original");
    }
}
```



Fresher Academy



- Gọi tới một object mà trạng thái object không cho chúng ta làm điều đó. Ví dụ:
Cố gắng viết vào readonly file thì trong trường hợp trạng thái file đó sẽ không cho phép thực hiện operation write

Use cases (2)

- An inappropriate call to an object is made, based on the object state.
 - One example might be trying to write to a read-only file. In cases where an object state does not allow an operation

```
class ProgramLog
{
    System.IO.FileStream logFile = null;
    void OpenLog(System.IO.FileInfo fileName, System.IO.FileMode mode) {}

    void WriteLog()
    {
        if (!this.logFile.CanWrite)
        {
            throw new System.InvalidOperationException("Logfile cannot be read-only");
        }
        // Else write data to the log and return.
    }
}
```



Fresher Academy



- Khi argument của một method causes exception thì trong trường hợp này nên bắt original exception và throw ra instance ArgumentException

Use case (3)

- When an argument to a method causes an exception
 - In this case, the original exception should be caught and an [ArgumentException](#) instance should be created

```
static int GetValueFromArray(int[] array, int index)
{
    try
    {
        return array[index];
    }
    catch (System.IndexOutOfRangeException ex)
    {
        System.ArgumentException argEx = new System.ArgumentException("Index is out of range", "index", ex);
        throw argEx;
    }
}
```



Fresher Academy



10 9 CustomExceptionVideo

- Custom Exception: Trong C# chúng ta có thể tạo ra customer exception bằng cách kế thừa exception class hoặc từ ApplicationException
- Microsoft: warning chúng ta nên tạo ra customException kế thừa trực tiếp từ exception thay vì sử dụng applicationexception. Bởi vì trong thực tế ko có quá nhiều lợi ích từ api exception
- Ví dụ:

Sample

```
5 references
public class CustomException : Exception
{
    0 references
    public CustomException()
        : base() { }

    0 references
    public CustomException(string message)
        : base(message) { }

    1 reference
    public CustomException(string format, params object[] args)
        : base(string.Format(format, args)) { }
}
```



Fresher Academy



Sample

```
5 references
public class CustomException: Exception
{
    0 references
    public CustomException()
        : base() { }

    0 references
    public CustomException(string message)
        : base(message) { }

    1 reference
    public CustomException(string format, params object[] args)
        : base(string.Format(format, args)) { }
}
```

```
    throw new CustomException("{0}'s age is {1}. It cannot be less than zero", args2);
```



Fresher Academy



10 12 ExceptionFilterVideo

- Exception filter là một trong những major feature mới của C# 6.0, nó cho phép chúng ta chỉ định condition trong catch block

Declaration

- Exception filters are one of the major new features of C# 6. They allow you to specify a condition on a catch block.

```
01 | static void Main()
02 | {
03 |     try
04 |     {
05 |         Foo.DoSomethingThatMightFail(null);
06 |     }
07 |     catch (MyException ex) when (ex.Code == 42)
08 |     {
09 |         Console.WriteLine("Error 42 occurred");
10 |     }
11 | }
```



Fresher Academy

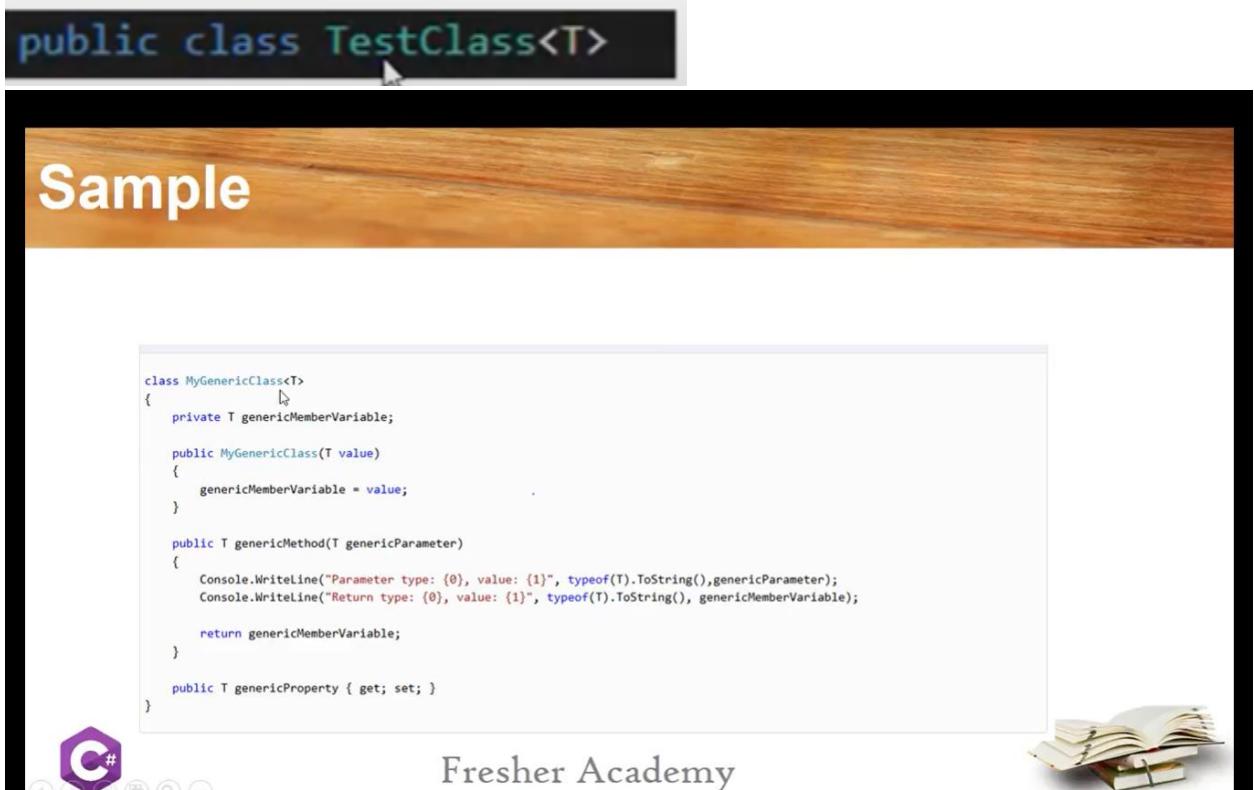


11 1 GenericVideo

- Generic: được thêm vào .NET Framework từ version 2.0 trở đi, và cho phép chúng ta delay việc chỉ định datatype của những thành phần trong một class hoặc trong một method cho đến khi sử dụng trong chương trình. Hay nói cách khác generic cho phép chúng ta viết ra những class hay method có thể làm việc với bất kể loại datatype nào
- Ưu điểm của generic:
 - Maximize code reuse, hỗ trợ type safety và performance
 - Use case phổ biến nhất của generic là tạo ra collection classes
 - Tạo ra generic ở trên interface, classes, method, event và trên delegate
 - Thông tin về generic data type có thể lấy ra tại thời điểm runtime bằng cách sử dụng reflection

11 2 GenericClass Video

- Generic class nó sẽ đóng gói các operation mà không chỉ định tới loại data type cụ thể nào
- Có thể tạo class trở thành generic class bằng cách thêm kí hiệu <T>



Sample

```

class MyGenericClass<T>
{
    private T genericMemberVariable;

    public MyGenericClass(T value)
    {
        genericMemberVariable = value;
    }

    public T genericMethod(T genericParameter)
    {
        Console.WriteLine("Parameter type: {0}, value: {1}", typeof(T).ToString(), genericParameter);
        Console.WriteLine("Return type: {0}, value: {1}", typeof(T).ToString(), genericMemberVariable);

        return genericMemberVariable;
    }

    public T genericProperty { get; set; }
}

```

 Fresher Academy 

Sample

```
MyGenericClass<int> intGenericClass = new MyGenericClass<int>(10);
.
int val = intGenericClass.genericMethod(200);
```

Parameter type: int, value: 200
Return type: int, value: 10

```
MyGenericClass<string> strGenericClass = new MyGenericClass<string>("Hello Generic World");
strGenericClass.genericProperty = "This is a generic property example.";
string result = strGenericClass.genericMethod("Generic Parameter");
```

Parameter type: string, value: Generic Parameter
Return type: string, value: Hello Generic World



Fresher Academy



11 4 GenericMethod Video

- Trong C# để biến method trở thành generic thì cần phải add thêm kí hiệu <T> và sau tên method trước dấu mở ngoặc để truyền parameter

Generic Methods

- In C#, to make your method generic, you need to add the <T> after the end of the method name and before the opening parenthesis of the method.

```
static void Swap<T>(ref T a, ref T b)
{
    T temp;
    temp = a;
    a = b;
    b = temp;
}
```



Fresher Academy



- Ví dụ:

Sample

```
int a = 40, b = 60;
Console.WriteLine("Before swap: {0}, {1}", a, b);
Swap<int>(ref a, ref b);
Console.WriteLine("After swap: {0}, {1}", a, b);

String x = "Hello", y = "Goodbye";
Console.WriteLine("Before swap: {0}, {1}", x, y);
Swap<string>(ref x, ref y);
Console.WriteLine("After swap: {0}, {1}", x, y);
```

```
file:///C:/Users/CuongNguyen/Docum
Before swap: 40, 60
After swap: 60, 40
Before swap: Hello, Goodbye
After swap: Goodbye, Hello
```



Fresher Academy



11 5 GenericInterface Video

- Generic interface cũng tương tự như generic class, chúng ta có thể define ra parameter T ở mức interface và method ở trong interface đó thì cũng có thể sử dụng parameter này, và những method ở trong class mà class implement generic interface cũng cần phải implement parameter T
- Ví dụ:

Sample

```
public interface IBook<T>
{
    1 reference
    void add(T book);
    1 reference
    void delete();
    1 reference
    T get();
}
```

```
public class Book<T> : IBook<T>
{
    T book;
    1 reference
    public void add(T book)
    {
        this.book = book;
    }
    1 reference
    public void delete()
    {
        this.book = default(T);
    }
    1 reference
    public T get()
    {
        return this.book;
    }
}
```

```
Book<string> stringBook = new Book<string>();
stringBook.add("Basic C#");
var book = stringBook.get();
Console.WriteLine(book);
```

```
file:///C:/Users/CuongN
Basic C#
```



Đang chđ www.youtube.com...

Fresher Academy



11 6 Collection Video

- Collection: trong C# cung cấp một số class đặc biệt để có thể lưu trữ giá trị hoặc object theo chuỗi cụ thể gọi là collection
- Có 2 loại collection là : non-generic collection và generic collection
- 2 loại collection này chia sẻ với nhau một số điểm chung sau
 - Tất cả đều implement loại interface IEnumerable bởi vậy chúng ta có thể sử dụng vòng lặp foreach để truy cập đến các item trong collection
 - Tất cả đều có thể copy tới mảng sử dụng method copyto
 - Tất cả index collection trong namespace System.collection đều sử dụng 0-indexed
 - Một số collection đều có capacity, một số khác có count, một số thì có cả 2. Capacity cho chúng ta biết collection đã chứa bao nhiêu item, còn count thì cho chúng ta biết hiện thời có bao nhiêu item
- Non-generic collections: nó sẽ store item giống như object có nghĩa rằng là item của thể loại non-generic collection có thể là bất kể loại data type nào. Chính vì vậy khi mà chúng ta retrieving, get lại item của loại collection này thì sẽ cần cast trở về một loại datatype cụ thể
- Trong bài này sẽ có 3 loại non-generic collection
 - ArrayList
 - HashTable
 - SortedList
- Generic collection : bên trong sẽ lưu trữ element dưới dạng array chính vì vậy phải chỉ định data type khi sử dụng generic collection và không cần phải boxing và casting item này. Trong bài này sẽ có 2 loại generic collection
 - List
 - Dictionary

11 7 ArrayList Video

- Trong C# thì ArrayList khá giống với array ngoại trừ chúng ta không cần phải chỉ định kích thước cho arraylist và không cần phải chỉ định datatype cho arraylist, khi add item cho arraylist thì tự động kích thước của arraylist được tăng lên
- Cú pháp

```
ArrayList myArrayList = new ArrayList();
```

- Properties của arraylist bao gồm:
 - Capacity: cho phép get hoặc set số lượng element ở trong một
 - Count: trả về số lượng element mà arraylist đang chứa.
- Common method
 - Add(): cho phép chúng ta thêm single element vào cuối của một arraylist
 - AddRange(): cho phép chúng ta add element từ một collection khác vào arraylist
 - Insert(): insert một element vào một vị trí mà ta chỉ định trong arraylist
 - insertRange: cho phép insert element từ một collection vào arraylist từ một vị trí mà ta chỉ định

- remove(): cho phép xoá element mà ta chỉ định từ arraylist
- removeRange(): xoá một range element ra khỏi arraylist
- Contains: kiểm tra element cụ thể có tồn tại trong arraylist hay không, nếu tồn tại trả về true và nếu không trả về false

11 9 SortedList Video

- Sortedlist collection cho phép chúng ta lưu trữ dưới dạng key value pairs, dưới dạng khoá và giá trị giống như hashtable, tên là sortedlist thì mặc định sắp xếp theo thứ tự tăng dần và sortedlist có thể implement cả 2 interface IDictionary & ICollection nên có thể truy cập vào item của sortedlist bằng key hoặc index
- Cú pháp:

```
SortedList sortedList1 = new SortedList();
```

- Thuộc tính cơ bản của sortedlist(common properties)
 - Capacity: cho phép chúng ta get và set số lượng item mà một sortedlist có thể chứa
 - Count: trả về tổng số item mà hiện giờ sortedlist đang chứa
 - Keys: trả về key mà sortedlist chứa
 - Values: trả về giá trị mà sortedlist chứa
- Method cơ bản của sortedlist(common method)
 - Add: thêm item với key và value vào trong sortedlist và tự động sortedlist sẽ order vào đúng vị trí của nó
 - Remove: xoá item ra khỏi sortedlist dựa trên key mà chúng ta truyền vào
 - Clear: xoá tất cả item trong sortedlist
 - Contains/containskey: kiểm tra xem key có tồn tại trong sortedlist hay không
 - Containsvalue: kiểm tra xem value có tồn tại trong sortedlist hay không.

11 11 HashTableVideo

- hashTable: trong C# sẽ cho phép chúng ta có thể lưu dưới dạng key-value pairs(dạng khoá và giá trị). Bên trong hashtable sẽ băm khoá của chúng ta thành hashcode và hashcode này sẽ được lưu ở different bucket vì vậy khi lấy ra một giá trị thì thực ra hash table sẽ match hashcode đó với key của chúng ta và sẽ tăng performance lúc tìm kiếm
- Cú pháp:

```
Hashtable ht = new Hashtable();
```

- Thuộc tính cơ bản(common Properties):
 - Count : trả lại tổng số cặp key-value mà hashtable đang chứa, hashtable không có thuộc tính capacity
 - Keys: trả lại tập hợp mà hashtable đang có dưới dạng ICollection
 - Values: để lấy lại tập hợp value trong hashtable
- Method cơ bản của hashtable(common methods)
 - Add: cho phép chúng ta add item dưới dạng key value vào trong hashtable

- Remove: cho phép xoá item với key chúng ta chỉ định từ hashtable
- Clear: xoá tất cả item trong hashtable
- Containskey: kiểm tra hashtable có chứa key chúng ta truyền vào hay không
- containsValue: kiểm tra value truyền vào có tồn tại hay không

11 13 DictionaryVideo

- Dictionary: trong C# giống quyền từ điển tiếng anh, bao gồm những từ và giải nghĩa cho từ đó thì tương tự như vậy dictionary là tập hợp của key và value, trong đó có thể key là từ và value là giải nghĩa
- Dictionary là loại generic collection, chính vì vậy chúng ta cần sử dụng là <TKey, TValue>, tất cả generic collection đều được include trong namespace gọi là system.collection.Generic
- Cú pháp

```
IDictionary<int, string> dict = new Dictionary<int, string>();
```

//or

```
Dictionary<int, string> dict = new Dictionary<int, string>();
```

- Thuộc tính cơ bản của Dictionary(Common Properties)
 - Count: trả về số lượng element đang tồn tại trong dictionary đó, ngoài ra có thể lấy ra danh sách <TKey, TValue>
- Phương thức của dictionary(Common method)
 - Add: cho phép thêm item vào trong dictionary đó
 - Remove: cho phép xoá element với key chỉ định
 - Clear : xoá tất cả item trong dictionary
 - ContainsKey/ContainsValue: kiểm tra xem value truyền vào có tồn tại hay không

11 15 ChoosingCollectionVideo

Chọn collection phù hợp trong C#

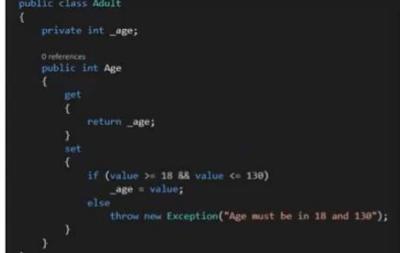
- Non-generic collection: cho phép store item dưới dạng object ở bất kì loại datatype nào. Trong Non-generic collection có 3 loại:
 - ArrayList: hoạt động như mảng cho phép chúng ta lưu danh sách của item, chẳng hạn lưu danh sách của product
 - Hashtable: lưu trữ dưới dạng key-Value
 - SortedList: muốn lưu trữ dưới dạng Key-Value nhưng lại muốn key được sắp xếp thì không phải làm nhiệm vụ sorted hay compare thì nên sử dụng sortedlist
- List thì giống arraylist
- Dictionary giống hashtable

AnnotationVideo

- Annotation: là chuỗi của attribute mà chúng ta có thể thêm vào trong một class để apply validation rules tới class của chúng ta
- Ví dụ:

Annotation

• Contains a series of attributes that you can add to your classes to apply validation rules to your classes







- Về cơ bản data annotations chỉ là thuộc tính tức là chỉ là metadata, tự thân nó sẽ không validate để xem giá trị gán vào có là validate hay không mà để kiểm tra xem là thuộc tính của chúng ta có được valid hay không thì cần sử dụng validator class

Annotation (2)

• Data annotations are just attributes or in other words they are meta-data. They themselves do not do anything. So, we need to run the validator class to check for the objects validations.

• Data Annotation validators can only be applied to properties

```
Adult Mike = new Adult();
Mike.Age = 16;
ValidationContext context = new ValidationContext(Mike, null, null);
List<ValidationResult> results = new List<ValidationResult>();
bool valid = Validator.TryValidateObject(Mike, context, results, true);

if (!valid)
{
    foreach (ValidationResult vr in results)
    {
        Console.WriteLine(vr.ErrorMessage);
    }
}
```





Anonymous Method Video

- Anonymous method: là phương thức nặc danh, có nghĩa là có thể tạo ra phương thức mà không phải chỉ định tên. Trong C# để tạo ra một phương thức nặc danh thì cần sử dụng từ khoá delegate

Definition

- An anonymous method is a method without a name. Anonymous methods in C# can be defined using the delegate keyword

```
public delegate void Print(int value);

static void Main(string[] args)
{
    Print print = delegate(int val) {
        Console.WriteLine("Inside Anonymous method. Value: {0}", val);
    };
    print(100);
}
```



Fresher Academy



- Phương thức nặc danh có đặc tính có thể access tới biến mà được khai báo bên ngoài nó

Some features

- Anonymous methods can access variables defined in an outer function.

```
public delegate void Print(int value);

static void Main(string[] args)
{
    int i = 10;

    Print print = delegate(int val) {
        val += i;
        Console.WriteLine("Anonymous method: {0}", val);
    };
    print(100);
}
```



Fresher Academy



- Tuy nhiên biến được khai báo bên trong phương thức nặc danh thì bên ngoài không access được
- Thông thường sử dụng phương thức nặc danh để event handling

Some features

- A variable declared inside the anonymous method can't be accessed outside the anonymous method.
- We use anonymous method in event handling.

```
saveButton.Click += delegate(Object o, EventArgs e)
{
    System.Windows.Forms.MessageBox.Show("Save Successfully!");
};
```



Fresher Academy



AnonymousTypeVideo

- C# cung cấp cho chúng ta một cách thức tiện lợi để có thể đóng gói một số thuộc tính readonly vào trong một object mà không cần phải chỉ định kiểu cho object đó

Anonymous Type

- Anonymous types provide a convenient way to encapsulate a set of read-only properties into a single object without having to explicitly define a type first, and we use var to hold the reference of anonymous types

```
var myAnonymousType = new { firstProperty = "First",
                           secondProperty = 2,
                           thirdProperty = true
};
```



Fresher Academy



- Phạm vi của anonymous type: cũng giống như biến local thông thường và chúng ta chỉ có thể sử dụng anonymous type trong method mà chúng ta đã defined, tuy nhiên khác với biến thông

thường là có thể truyền nó như parameter vào một cái method. Anonymous type không thể truyền parameter vào method, ngoại trừ method đó sử dụng từ khoá dynamic

Anonymous Type Scope

- An anonymous type will always be local to the method where it is defined. Usually, you cannot pass an anonymous type to another method unless use dynamic

```
static void Main(string[] args)
{
    var myAnonymousType = new
    {
        firstProperty = "First Property",
        secondProperty = 2,
        thirdProperty = true
    };

    DoSomething(myAnonymousType);
}

static void DoSomething(dynamic param)
{
    Console.WriteLine(param.firstProperty);
}
```



Fresher Academy



- Anonymous types hay được sử dụng trong linq

Anonymous Types with a LINQ query

```
IList<Student> studentList = new List<Student>() {
    new Student() { StudentID = 1, StudentName = "John", age = 18 },
    new Student() { StudentID = 2, StudentName = "Steve", age = 21 },
    new Student() { StudentID = 3, StudentName = "Bill", age = 18 },
    new Student() { StudentID = 4, StudentName = "Ram", age = 20 },
    new Student() { StudentID = 5, StudentName = "Ron", age = 21 }
};

var studentNames = from s in studentList
    select new { StudentID = s.StudentID,
        StudentName = s.StudentName
    };
```



Fresher Academy



DelegateVideo

- Delegate: một function thì có thể có một hoặc nhiều parameter thậm chí là không có parameter nào và parameter có thể thuộc bất kì loại datatype nào. Tuy nhiên nếu trong trường hợp muốn truyền một function giống parameter vào một function khác thì làm sao để làm dc việc đó.
- Trong c# nếu bạn muốn khai báo một con trỏ hàm tức là muốn truyền một function giống parameter tới function khác thì chúng ta sử dụng từ khoá delegate. Thì delegate là kiểu references data type mà nó giữ references tới method đó

Delegate

- A function can have one or more parameters, but what if you want to pass a function itself as a parameter?
- A delegate is like a pointer to a function. It is a reference data type and it holds the reference of a method.

```
public delegate void Print(int value);
```



Fresher Academy



- Khi biến Print thành delegate thì nó có thể trả về bất kì method nào nhận parameter là kiểu int mà trả lại là kiểu void
- Ví dụ:

Sample

```
class Program
{
    // declare delegate
    public delegate void Print(int value);

    static void Main(string[] args)
    {
        // Print delegate points to PrintNumber
        Print printDel = PrintNumber;
        printDel(10000);
        printDel(200);

        // Print delegate points to PrintMoney
        printDel = PrintMoney;

        printDel(10000);
        printDel(200);
    }

    public static void PrintNumber(int num)
    {
        Console.WriteLine("Number: {0,-12:N0}", num);
    }

    public static void PrintMoney(int money)
    {
        Console.WriteLine("Money: {0:C}", money);
    }
}
```

```
Number: 10,000
Number: 200
Money: $ 10,000.00
Money: $ 200.00
```



Fresher Academy



- Để thực thi delegate là references tới method nên chúng ta có thể invoked một delegate giống như invoked một method

Invoking Delegate

- The delegate can be invoked like a method because it is a reference to a method

```
Print printDel = PrintNumber;
printDel.Invoke(10000);

//or
printDel (10000);
```



Fresher Academy



EventVideo

- Cho ví dụ: Microsoft khởi động một sự kiện dành cho lập trình viên, giới thiệu về tính năng mới của sản phẩm thì khi Microsoft làm ra sản phẩm như vậy họ cần phải thông báo cho dev biết về sự kiện đó bằng cách gửi mail hoặc quảng cáo qua google hay facebook. Trong trường hợp này

chúng ta coi Microsoft là publisher và sau đó người ta phải notifies tới dev về sự kiện đó, khi dev biết về event thì sẽ phải đăng ký tức là chúng ta là subscribers của event và khi event hiện ra chúng ta sẽ phải tham dự(attend) hay nói cách khác là handle event đó.

Event

- For example, Microsoft launches events for developers, to make them aware about the features of new or existing products. Microsoft notifies the developers about the event by email or other advertisement options. So in this case, Microsoft is a **publisher** who launches (raises) an **event** and **notifies** the developers about it and developers are the **subscribers** of the event and attend (**handle**) the event.
- Events in C# follow a similar concept. An **event** has a **publisher**, **subscriber**, **notification** and a **handler**. Generally, UI controls use events extensively. For example, the button control in a Windows form has multiple events such as click, mouseover



Fresher Academy



- Trong C# thuật ngữ event cũng tương tự như phía trên cũng sẽ có publisher(là nơi lưu và write ra event), cũng có subscriber để register đến event đó. Có notification publisher notify đến subscriber và nó cũng có handle để subscriber có thể handle event
- Event đơn giản chỉ là đóng gói của delegate

ExtensionMethodVideo

- Extension method: cho phép chúng ta thêm các method vào một kiểu data type đã tồn tại mà không cần phải tạo ra class mới kế thừa từ data type, datatype có thể là kiểu int, string.v.v.v. Chúng ta không cần phải recompiling hoặc modify sources code của type original

```
public static int WordCount(this string str)
{
    string[] userString = str.Split(' ');
    int wordCount = userString.Length;
    return wordCount;
}
```

```
var userSentance = "Xin chao cac ban";
int totalWords = userSentance.WordCount();
Console.WriteLine(totalWords);
```

4

Lambda Expression Video

- Lambda expression cho phép chúng ta viết ra những method một cách tiện lợi
- Ví dụ:

The screenshot shows a video player interface. At the top, there is a title bar with the text 'Lambda Expression'. Below the title bar, there is a large, light-colored wooden background image. In the center of this image, there is some code. The code is partially visible, showing:
delegate(Student s) { return s.Age > 12 && s.Age < 20; };

s => s.Age > 12 && s.Age < 20

At the bottom left of the slide, there is a small logo for C# with the text 'C#'. At the bottom right, there is a logo of an open book with the text 'Fresher Academy'.

Steps

```
delegate(Student s) { return s.Age > 12 && s.Age < 20; };
© TutorialsTeacher.com
deleXate(SXudent s){ return s.Age > 12 && s.Age < 20; };
(s) => { return s.Age > 12 && s.Age < 20; };

(s) => X return X s.Age > 12 && s.Age < 20 X X;
© TutorialsTeacher.com
(s) => s.Age > 12 && s.Age < 20;
            3 - Remove Parenthesis around parameter if there
            is only one parameter
s => s.Age > 12 && s.Age < 20;
```



Fresher Academy



OptionalAndNamedParamVideo

- Optional and Named Parameter

Optional and Named Parameter

- It comes from C# 4.0

```
public static double CurrencyExchange(double amount, double rate)
{
    return (amount * rate);
```

```
CurrencyExchange(1000, 22.5);
```

```
public static double NewCurrencyExchange(double amount, double rate = 22.5)
{
    return (amount * rate);
```

```
NewCurrencyExchange(1000);
```

```
NewCurrencyExchange(rate: 22.8, amount: 1000);
```



Fresher Academy



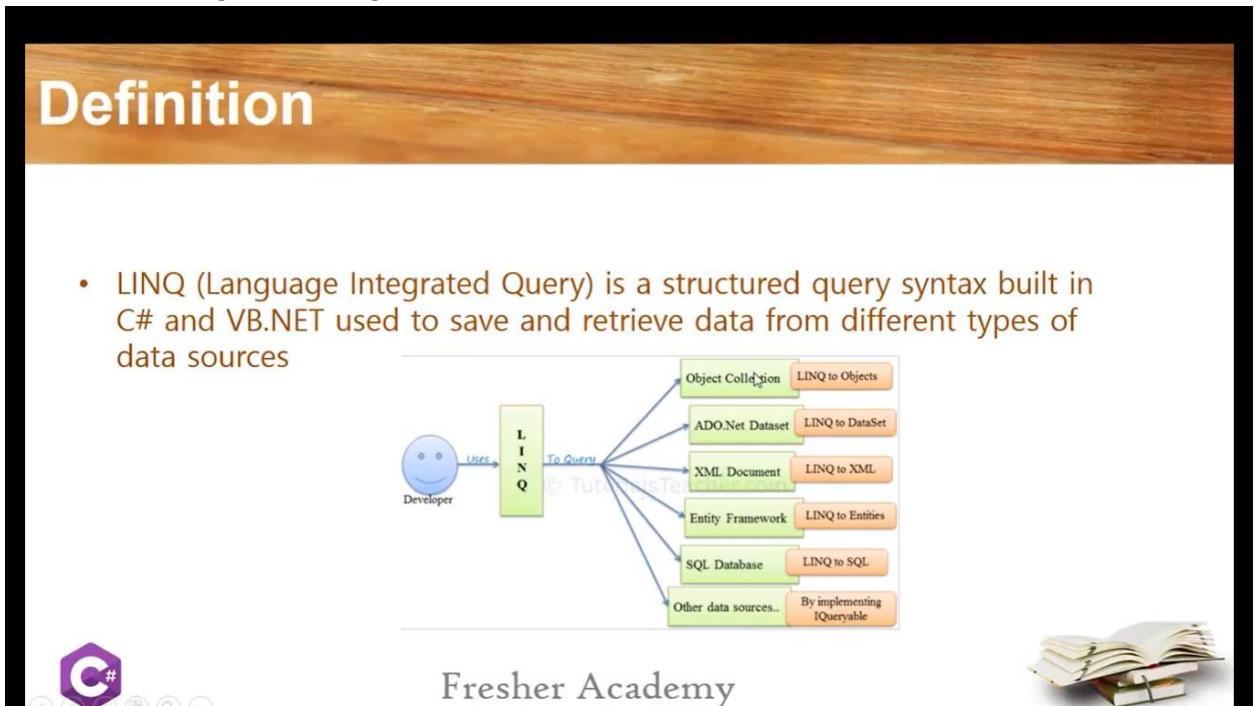
PrivateVsPublicAssemblyVideo

- Private Assembly: Trong C# có 3 loại assembly
 - Private Assembly: chỉ sử dụng bởi một ứng dụng

- Private Assembly: sẽ được stored trên directory hoặc sub-directory
- Private Assembly không cần strong name
- Public Assembly:
 - Public Assembly: có thể sử dụng bởi multiple application(viết xong project có thể biến nó thành một assembly)
 - Public Assembly: được stored trên GAC(Global Assembly Cache)
 - Public Assembly: đòi hỏi phải có Strong name

Query Expression Video

- LinQ(Language Integrated Query) là cú pháp cho phép chúng ta truy vấn dữ liệu có cấu trúc nó được built in trong C# để chúng ta có thể save nhận dữ liệu từ data source khác nhau



Why LINQ?

```
class Program
{
    static void Main(string[] args)
    {
        Student[] studentArray = {
            new Student() { StudentID = 1, StudentName = "John", Age = 18 },
            new Student() { StudentID = 2, StudentName = "Steve", Age = 21 },
            new Student() { StudentID = 3, StudentName = "Bill", Age = 25 },
            new Student() { StudentID = 4, StudentName = "Ram", Age = 28 },
            new Student() { StudentID = 5, StudentName = "Ron", Age = 31 },
            new Student() { StudentID = 6, StudentName = "Chris", Age = 17 },
            new Student() { StudentID = 7, StudentName = "Rob", Age = 19 },
        };

        Student[] students = new Student[10];
        int i = 0;

        foreach (Student std in studentArray)
        {
            if (std.Age > 12 && std.Age < 20)
            {
                students[i] = std;
                i++;
            }
        }
    }
}
```



Fresher Academy



- Cú pháp query linq có 2 cách:
 - Query syntax

```
// LINQ Query Syntax
var result = from s in stringList
             where s.Contains("Tutorials") |
             select s;
```

- Method syntax:

```
// LINQ Query Syntax
var result = stringList.Where(s => s.Contains("Tutorials"));
```

NullConditionalVideo

Problem

```
Console.WriteLine(customers.Length);
Console.ReadKey();
```

An unhandled exception of type 'System.NullReferenceException' occurred in ConsoleApplication1.exe
Additional information: Object reference not set to an instance of an object.

Troubleshooting tips:
Check to determine if the object is null before calling the method.
Use the "new" keyword to create an object instance.
Get general help for this exception.

```
Console.WriteLine(customers[0]);
Console.ReadKey();
```

An unhandled exception of type 'System.NullReferenceException' occurred in ConsoleApplication1.exe
Additional information: Object reference not set to an instance of an object.

Troubleshooting tips:
Check to determine if the object is null before calling the method.
Use the "new" keyword to create an object instance.
Get general help for this exception.

```
if (customers != null)
{
    Console.WriteLine(customers.Length);
    Console.WriteLine(customers[0]);
}
Console.ReadKey();
```



Fresher Academy

- Từ C# 6.0 trở đi đã có operator đặc biệt gọi là null conditional, sẽ giúp cho chúng ta không phải viết code dài dòng và lặp đi lặp lại nữa, và sẽ có 2 kiểu cú pháp đó là
 - ? . hay còn gọi là elvis operator
 - ? [

Null-conditional Operators

- This is new from C# 6.0, helps in performing null, so that we don't need to write long repetitive code to handle it. There is 2 syntax
 - ? . (also called elvis operator)
 - ? [

```
if (customers != null)
{
    Console.WriteLine(customers.Length);
    Console.WriteLine(customers[0]);
}
Console.ReadKey();
```

```
Console.WriteLine(customers?.Length);
Console.WriteLine(customers?[0]);
Console.ReadKey();
```



Fresher Academy

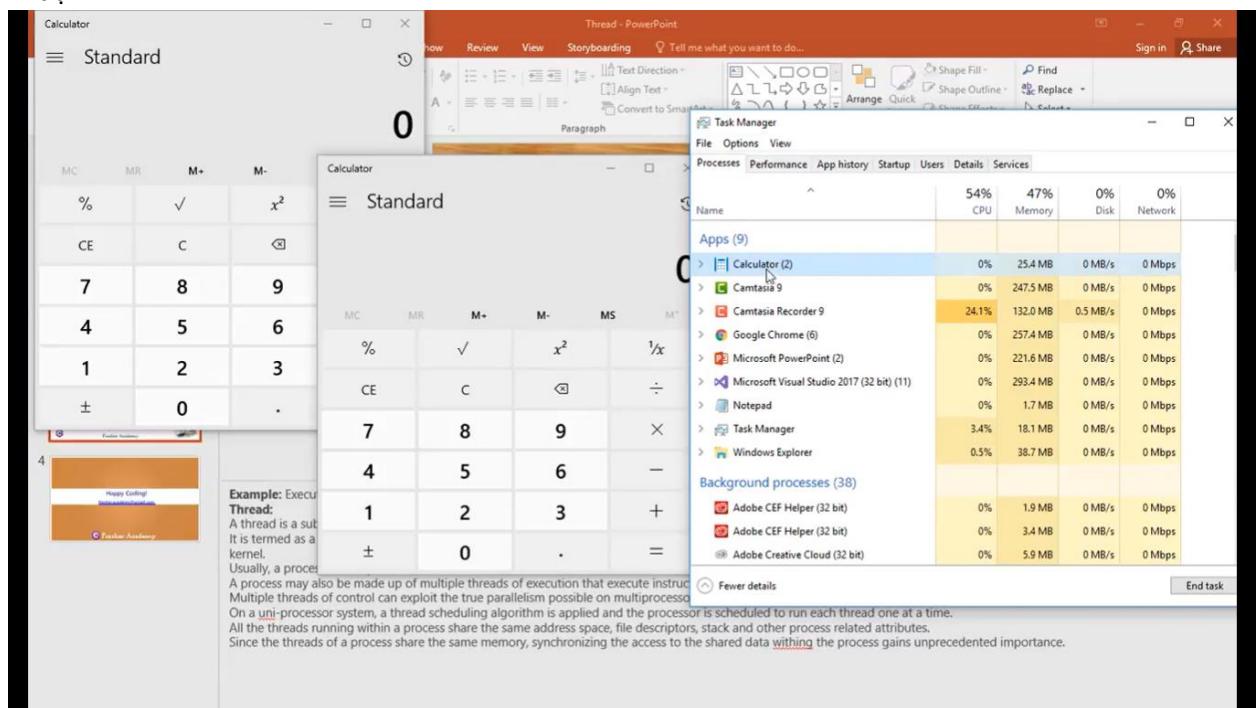
12 1 Concurrency Video

- Concurrency: là cố gắng làm nhiều hơn một việc tại một thời điểm, chẳng hạn chúng ta có:

- Ứng dụng end-user thì ứng dụng này có thể sử dụng concurrency để phản hồi lại những user input mà khách hàng nhập lên form trong khi nó vẫn đang viết request đó xuống database
- Ứng dụng server nó sẽ cần phản hồi rất nhiều request client gửi lên thì server application có thể sử dụng concurrency để phản hồi yêu cầu thứ hai trong khi vẫn đang cố gắng hoàn thành phản hồi đầu tiên

12 1 ThreadVideo

- Process and Thread
- Tiến trình(Process) là chương trình đang chạy trên máy tính, và một chương trình thì có thể associated với nhiều process. Ví dụ có thể mở ra chương trình caculator trên máy tính và mở nhiều instances của calculator và mỗi instances đấy tương đương với một process
- Ví dụ:



- Thread thực ra chính là sắp xếp, tức là tập con của process. Có nghĩa rằng là trong một process thì có một hoặc nhiều thread và các thread này được đặt lịch để chạy chương trình của chúng ta. Thread có thể hiểu là đơn vị cơ bản mà hệ điều hành sẽ allocate resource để có thể chạy ứng dụng
- Trong C# những chương trình như Console được viết từ đầu đến giờ thì tự động thằng CLR và hệ điều hành sẽ tạo ra thread mặc định cho chúng ta và những chương trình đó chạy trên thread

12 3 SynchronousVsAsyncVideo

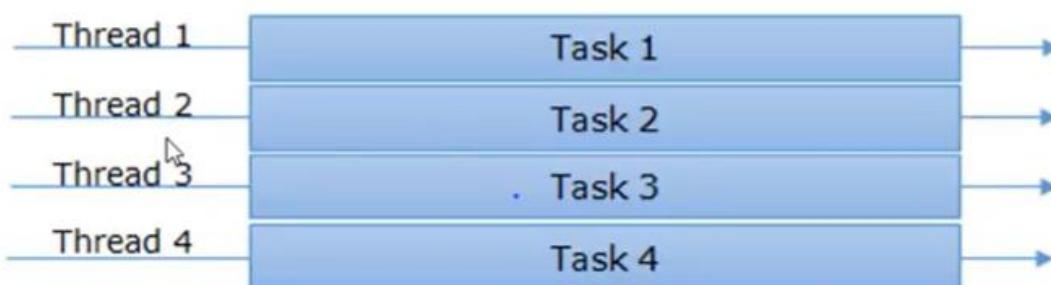
- MultiThreading: đến giờ chúng ta vẫn đang làm việc với console application và sử dụng thread để thực thi chương trình hay còn gọi là single thread. Tuy nhiên trong C# hay .NET Framework cho phép sử dụng multithreading(đa chương trình). Multithreading gọi là dạng thức của concurrency
- Trong một máy tính thì CPU có rất nhiều core và sẽ là không hợp lý nếu chương trình làm rất nhiều việc trong khi chỉ sử dụng một core để làm việc trong khi core khác thì idle. Thường thì chúng ta

có thể split chương trình thành nhiều tác vụ và tác vụ đấy có thể chạy trên nhiều thread khác nhau và thread đấy có thể chạy trên nhiều core khác nhau

- Synchronous Programming Model: (Lập trình đồng bộ) trong mô hình lập trình này thì một thread được assigned cho một task thì nó sẽ bắt đầu làm việc trên task đó và chỉ khi task đó hoàn thành thì thread đấy mới available và làm việc với task tiếp theo. Mô hình lập trình đồng bộ có thể sử dụng single thread hoặc multiple thread
- Trong trường hợp sử dụng Single Thread: thì có nghĩa là nó chỉ sử dụng một thread để làm việc với những tác vụ chương trình

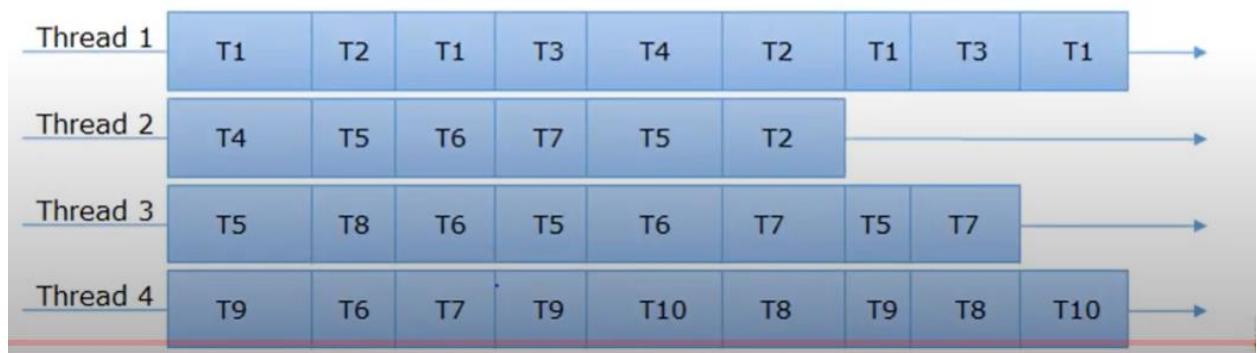


- Lập trình đồng bộ có hỗ trợ làm việc trên multiple thread: có nghĩa là chúng ta sẽ sử dụng nhiều thread để làm việc song song với tác vụ trên máy



⇒ Một thread chỉ làm việc với một tác vụ tại cùng một thời điểm, chỉ khi nó hoàn thành tác vụ đấy thì nó mới available cho tác vụ tiếp theo

- Asynchronous Programming(lập trình bất đồng bộ): là một dạng thức của concurrency, trong lập trình bất đồng bộ một thread sẽ làm việc với nhiều tác vụ một lúc. Giả sử chúng ta có một thread và ta sẽ assign một tác vụ cho nó thì thread đấy sẽ thực hiện tác vụ đấy và nó không cần đợi khi tác vụ đấy xong, có thể pause giữa chừng để nó làm việc với tác vụ thứ 2 và save current state của tác vụ một lại



⇒ Trong lập trình bất đồng bộ một thread sẽ làm việc với multiple task tại cùng một thời điểm

12 4 ThreadClassVideo

Thread Class

```
Thread threadA = new Thread(new ThreadStart(ExecuteA));
1 reference
private static void ExecuteA()
{
    Console.WriteLine("Executing parameterless thread!");
}
```

```
Thread threadB = new Thread(new ParameterizedThreadStart(ExecuteB));
2 references
private static void ExecuteB(Object obj)
{
    Console.WriteLine("Executing thread with parameter \"{0}\"!", obj);
}
```

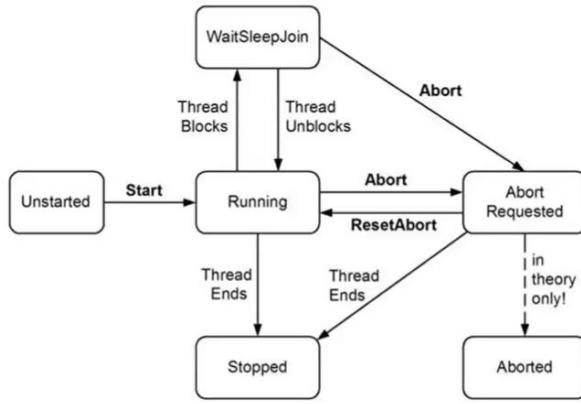


Fresher Academy



- Vòng đời của thread

Thread Life Cycle



Fresher Academy

- Properties của thread
 - **threadState**: chúng ta có thể lấy ra trạng thái của thread hiện tại
 - **Priority**: sử dụng multi thread để get hoặc set độ ưu tiên đó
 - **CurrentContext**: lấy được current context mà hiện thời thread đang chạy

- IsAlive: trả về true nếu thread được start và nó chưa bị stop, còn trong trường hợp khác là false
- IsBackground: get hoặc set xem thread có phải background hay không

12 5 ThreadClass2Video

- C# có 2 loại thread:
 - Foreground Thread: là cái tiếp tục chạy ngay cả khi chúng ta thoát chương trình và tiếp tục chạy
 - Background thread: là những thread bị terminate khi mà tất cả foreground thread bị đóng. Có nghĩa là ngay cả khi background thread chưa hoàn thành công việc nhưng mà foreground thread bị close rồi thì background thread cũng sẽ bị terminate. Thường thì chúng ta sẽ sử dụng Background thread cho tác vụ không quá quan trọng
 - Default khi tạo thread sẽ là foreground
- Main Thread: thread được thực thi đầu tiên trong process thì sẽ được gọi là main thread
 - Trong một chương trình C# khi mà bắt đầu chạy thì main thread sẽ được tự động tạo ra và những thread được tạo ra bằng cách sử dụng thread class
- Thread method:
 - Start(): start thread
 - Sleep(int): pause thread trong khoảng thời gian truyền vào(mili second)
 - Abort(): nó sẽ raises ra exception ThreadAbortException và nếu không sử dụng method reset abort() thì thread sẽ bị terminate
 - Join(): tất cả thread nằm sau phương thức join sẽ phải đợi cho đến khi thread đang gọi join hoàn thành thì nó mới được thực thi

12 7 TaskAsyncAwait Video

- Lập trình bất đồng bộ là một dạng thức của concurrency và để thực thi concurrency thì lập trình bất đồng bộ cần sử dụng khái niệm futures hay là callback để tránh việc sử dụng thread không cần thiết
- Future hoặc callback là trong lập trình bất đồng bộ có thể gọi tới method nhưng chúng ta không cần phải đợi đến khi method hoàn thành xong công việc thì mới thực thi tới method tiếp theo. Trong lập trình bất đồng bộ ta sẽ gọi tới method xong sẽ quên method đó đi, trong tương lai sau một khoảng thời gian method đó hoàn thành và trả lại giá trị thì lúc ta lại xử lý tiếp với method đó. Với việc trả lại giá trị thì trong .NET nó sẽ sử dụng Task hoặc Task<TResult>
- Trong C# 5.0 nó có thể đơn giản hóa lập trình bất đồng bộ cho chúng ta bằng cách sử dụng 2 từ khoá Async và await
- Từ khoá Async sẽ xác định một method của chúng ta là bất đồng bộ, ví dụ:

```
private static async void Method1()
{
    ...
}
```

- Asynchronous method sẽ trả lại 3 loại giá trị:
 - Void: không trả lại gì cả và trong lập trình bất đồng bộ khi mà sử dụng void có nghĩa là “fire and forget” gọi tới method đó và quên luôn (không cần biết là có hoàn thành hay không)
 - Task: không trả lại giá trị gì nhưng cho phép caller tức là thằng container gọi tới method biết rằng là method đó có finish hay không. Tức là trong tương lai method1 hoàn thành thì thằng caller biết là đã finish để xử lý tiếp
 - Task<T>: trả lại giá trị
- Await: đợi, tức là gọi đến function Asynchronous bằng cách sử dụng await

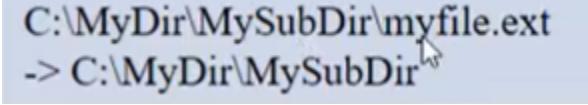
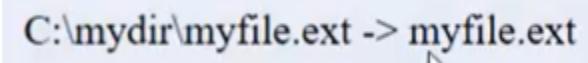
```
await Method1();
```

9 0 FileStreamPathDirectory Video

- I/O: trong .NET Framework nó cung cấp một namespace gọi là system.IO và namespace này chứa rất nhiều class cho phép chúng ta có thể đọc, ghi file hoặc data streams một cách dễ dàng
- File là file chứa dữ liệu như file word, excel và save nó lại thì được gọi là file. Bên trong file là tập hợp byte và những byte này sẽ được lưu ở trên ổ đĩa, một file thì luôn được chỉ định tên và đường dẫn đến file đó và khi chúng ta viết một chương trình mở file đó ra cho nhiệm vụ đọc, ghi thì file đó sẽ biến thành stream
- Stream(Luồng): nó đơn giản là sequence của byte, tức là tập hợp của các byte cho phép chúng ta có thể đọc, và ghi những byte đó vào trong ổ đĩa hoặc memory
- Directory(thư mục): là container cho các file, file khi được tạo ra sẽ cần nơi lưu vào và nơi lưu đấy sẽ nằm trong thư mục
- Path: cung cấp đường dẫn đến file hoặc thư mục

9 1 Path Video

- Một path chỉ định ra location vào file và directory của chúng ta, path đơn giản là một chuỗi, là một string instance. Path class sẽ cung cấp cho chúng ta method và properties để process trên string instances đó, và path class sẽ hoạt động cross-platform (chạy được với hệ thống windows và linux)
- Path gồm có 2 loại là đường dẫn tuyệt đối và đường dẫn tương đối
 - C:\Windows\System32 là đường dẫn tuyệt đối
 - Windows\System32 là đường dẫn tương đối, nếu mình đang ở ổ C và copy đường dẫn và dán lại thì có thể đến đúng system32 tuy nhiên nếu đang ở C:\Program files thì paste thì sẽ không đi được đến system32 bởi vì trong Program file không có system32 (phụ thuộc vào điểm đứng)

- Phương thức cơ bản của path class(Method)
 - GetDirectoryName: trả lại tên của directory cho một path mà chúng ta truyền vào, ví dụ:

 - GetFileName: trả lại cả tên file và phần mở rộng của file đó cho đường dẫn mà chúng ta truyền vào, ví dụ:

 - GetFileNameWithoutExtension: không trả lại phần extension mà chỉ trả lại file name
 - GetFullPath: trả lại đường dẫn tuyệt đối cho path mà ta truyền vào
 - GetExtension: chỉ trả lại extension mà không trả lại file name

- 9 3 DirectoryInfo Video

- DirectoryInfo class cũng là class của system.IO. Directory cung cấp cho chúng ta những method có thể tạo, move, có thể duyệt các directory hoặc subdirectories
- Để tạo ra instance của directoryinfo class thì chúng ta sử dụng cú pháp sau
 - DirectoryInfo di = new DirectoryInfo(path)
- Common properties Directory
 - Exists: cho biết thư mục đấy đã tồn tại hay chưa, nếu trong trường hợp chưa tồn tại sẽ trả lại là false và ngược lại
 - Name: trả lại tên của directory
 - Parent: trả lại parent directory của thư mục hiện tại
 - FullName: trả lại full path của directory
- Common methods của directory
 - Create: tạo ra directory
 - Delete(Boolean): xoá đi directory hiện tại, trong trường hợp directory chứa thư mục con hoặc file truyền vào biến Boolean là false thì nó sẽ báo lỗi(không empty và không xoá được), còn nếu muốn xoá được thì phải trả về true
 - GetDirectories(String, SearchOption): trả lại danh sách những directory nằm trong directory hiện tại
 - GetFiles(String, SearchOption): trả lại danh sách các file nằm trong file directory

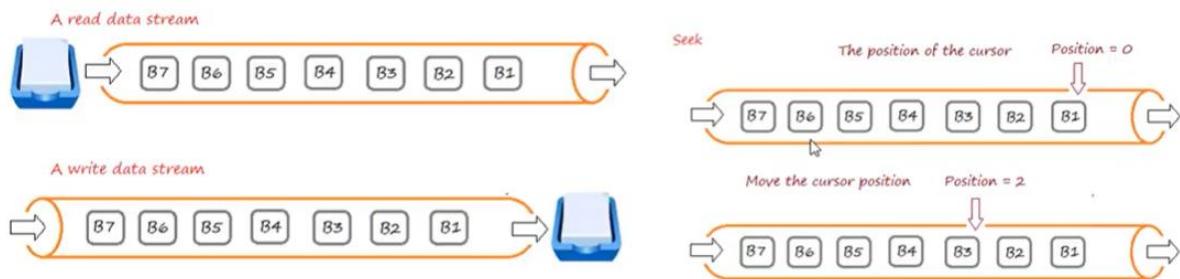
9 5 FileInfo Video

- FileInfo class cũng là một class của namespace system.IO, nó sẽ cung cấp cho chúng ta những properties và những method cho việc tạo, moving, copy, delete và mở file
- File là tập hợp của những byte và khi file được mở ra cho nhiệm vụ đọc hoặc ghi thì nó sẽ trở thành stream(luồng) và fileinfo class sẽ hỗ trợ tạo object thuộc dạng filestream
- Cú pháp tạo:
 - FileInfo fs = new FileInfo(path)
- Common properties()
 - Name: trả lại tên file
 - FullName: trả lại full đường dẫn đến file đó

- Length: trả lại size của byte theo file
- Extension: trả lại phần mở rộng của file
- IsReadOnly: cho biết file có readonly hay không
- Exists: cho biết file có tồn tại hay không, trả về true là tồn tại và ngược lại
- Common method
 - Open(FileMode, FileAccess): cho phép open một file để đọc, ghi hoặc là vừa đọc vừa ghi. Khi sử dụng method open thì sẽ truyền vào 2 parameter đó là FileMode và FileAccess>FileMode là sẽ chỉ định: muốn open hay muốn append vào cuối file, FileAccess: muốn đọc, ghi hay vừa đọc vừa ghi
 - Create(): tạo file trên đĩa
 - MoveTo(string destFile): cho phép chuyển file tới vị trí mới và có thể đổi tên file
 - Delete(): xoá file

9 6 Stream Video

- Stream class: file class là tập hợp các byte và khi mở file ra để đọc và ghi thì nó sẽ biến thành luồng của byte. Và system.IO nó có một abstract class gọi là stream class cung cấp cho ta những properties và những method cơ bản để có thể đọc ghi mảng của những byte này



- Common properties(thuộc tính của luồng)
 - CanRead/CanWrite/CanSeek: hỗ trợ cho việc đọc, ghi hay việc tìm kiếm
 - Length: trả về độ dài stream theo byte
 - Position: có thể get set vị trí con trỏ trong luồng đó
- Common methods
 - Write(Byte[], Int32, Int32): luồng cho phép chúng ta có thể lưu một mảng những byte vào trong ổ đĩa và cung cấp thêm 2 parameter kiểu int32 là để chỉ vị trí của byte mà chúng ta muốn ghi vào và độ dài số lượng byte muốn ghi
 - Read(Byte[], Int32, Int32): cho phép đọc một mảng các byte và cung cấp đọc từ vị trí nào tùy theo số lượng byte muốn đọc
 - Seek(Int64, SeekOrigin): cho phép set lại position trong luồng

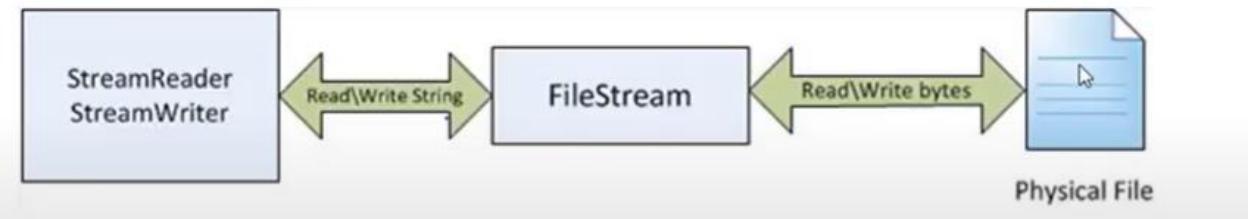
9 7 FileStream Video

- FileStream: khi mở file cho nhiệm vụ đọc và ghi thì file đó sẽ trở thành stream và filestream là một trong những luồng. FileStream sẽ support tác vụ đọc và ghi những byte từ file vật lý dù file đó là file txt, .exe, .jpg.v.v.
- File stream sẽ hỗ trợ đọc và ghi đồng bộ và bất đồng bộ
- Common properties

- CanRead, CanWrite: cho phép stream hiện tại cho phép đọc hay không
- Length: trả về kích cỡ dưới dạng byte của một stream
- IsAsync: file stream đã được mở ở dạng đồng bộ hay bất đồng bộ
- Common method
 - Write(Byte[], Int32, Int32): ghi block hoặc byte vào trong ổ đĩa
 - Read(Byte[], Int32, Int32): cho phép đọc block hoặc byte từ luồng và ghi dữ liệu đã vào buffer

9.9 StreamReader StreamWriter Video

- StreamReader/StreamWriter class: .NET Framework cung cấp 2 abstract class là TextReader và TextWriter cho phép ta có thể đọc và ghi text file, StreamReader/StreamWriter chính là implementation của TextReader/Writer



- Cú pháp:
 - StreamReader sr = new StreamReader(path);
 - StreamWriter sr = new StreamWriter (path);
- Common Methods
 - StreamReader
 - Read(): đọc các kí tự trong stream và trả về vị trí của kí tự đó, trả lại kiểu Int32. Còn khi đã đọc hết luồng thì trả về -1
 - ReadLine(): trả về dòng tiếp theo của stream đó, và trả lại null nếu đã đọc hết
 - Peek(): trả lại kiểu số cho kí tự tiếp theo cần đọc và sẽ trả lại -1 nếu hết cái đọc
 - StreamWriter
 - Write(string): cho phép write string vào trong stream và từ stream đó tự động sắp xếp vào trong phisical
 - WriteLine(string): Viết một chuỗi vào trong Luồng và sẽ terminate bởi dấu xuống dòng