Dot Net

Vikram Salakhe

M T W T F S S
Page No.:
Date: 18/07/22
YOUVA

Before .Net

FrontEnd ⟷ MIDDLE TIER (COM) ⟷ Database (ANY)
VC++/VB6/ASP

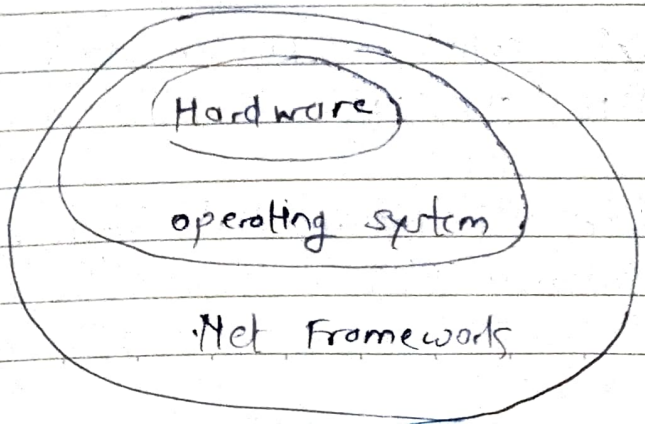
VC++/VB6 → Desktop applications.
ASP → web based applications
COM:
(component object model)

COM → mainly versioning & deployment
DLL Hell!

.Net Framework come because of these problems.

.Net Features
1) OO code
2) Multiple Language (over 50)
3) Multiple platforms
4) Multiple project types (eg web based, desktop based)
5) Better security
6) Improved performance.

.Net Framework



web application
windows form
Console apps
Web Services
WCF -
WPF - Windows Presentation foundation
Workflow
ASP.Net MVC
.Net core
Xamarin (ज़ामरीन)
window services
Web API

.Net Base class library
System.DII , System.Data.DII , System.Xml.DII etc

Assembly - (like java package)

Source code
(Any.Net language)
C#, Vb.net
We are using C#

↓ compile

Assembly
(Exe / DII) extension
Byte code (Intermediate language) just like Byte code in java
(MSIL / CIL / IL) IL format

Microsoft intermediate language
common IL

exe entry point
you can run assembly directly
dll → you cannot directly run dll

Native code - code that runs directly in machine format.

To run the code CLR is used

Common Language Runtime (CLR)

JIT compilation → Just In Time
→ convert IL code into native code
jitter is short form for JIT

Q. Why called just in Time?

```
main()
{
    func1();
    func1();
}
func1(){}
func2(){}
func3(){}
:
funcN(){}
```

It take only main function.
convert IL to native code

then func1 will convert IL to native code

Native code → lowest possible fractures
At Runtime
There multiple JIT compiler

Q. .Net code or native code which is faster
.Net code is faster.

898 1272 7028
758450

M T W T F S S
Page No.:
Date:
YOUVA

CLR
→ JIT compilation
→ Memory management
→ Garbage collection. → free memory
↳ Garbage memory
no longer reference

```
Main ()
{
    class O;
    O = new class();
}   O = null
```

Garbage collector is responsible for

when memory is going low CLR is calling garbage collection.

destructor ?
use to release the resources
when memory is released then destructor is called
We don't know when destructor is called.

You should not write any code for destructor in .Net.

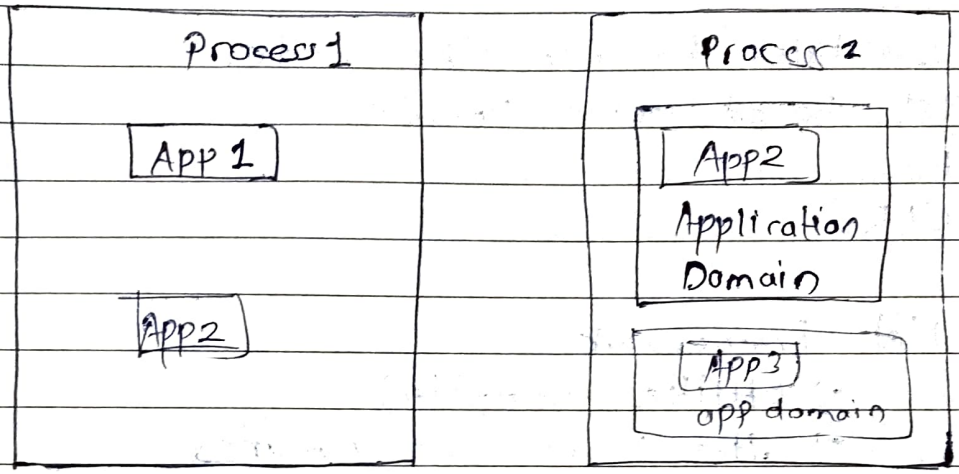
Can we write our code in destructor → No

function called dispose

Non deterministic finalization
↓
which cannot free memory determined.

Generations in garbage collection.

→ App Domain Management.



Highly unstable

Creating app domain & managing app domain
app domain restricted to that application.

→ Common Languge specification → CLS
→ Common Type System → CTS

CLS → Set of rules that all .net langeges must follow
.net must support only single inheritance.

CTS :
pre .Net .Net
Common type system datatype
compile convert int Int32
C#
int Doomething (int i) --> Int 32 i; →{
Vb.net
Dim i as Integer → Int32 i →{
i = Doomething (i)

→ Thread Management
→ Security Management.
→ Debugging
→ Exception Handling

Essentials to run the code
.Net Based class libraries
+
CLR

1) .Net Framework
   → Utilities (compiler)
   → .Net based class libraries
   → CLR

2) Project - Mono → for Linux
   .Mono is free & open source
   Mono → Spanish → meaning (Monkey)

3) .Net core → that works on other operating systems.
   → works on all platform

.Net used for windows
Mono used for Linux
Xamarin used for mobile application
.Net core → all platform.

.Net core

→ open source
→ cross platform
→ Light weight → supported files are there
→ Extensible

upto version 2.2 only support Asp.net Mvc and web Apis.
Version 3+ supports winforms & WPF also
Couse covers version 5 (.Net5)
.Net 6 released in Nov 2021

4.8 is last version in .Net framework
.Net core 5 → .Net 5 (means .Net core 5)
↓
.Net 6 (Nov 2021)

Managed code vs Unmanaged Code
→ Managed code is run by CLR
Before .Net there is unmanged code →
e.g. window DLLs (PInvoke) & COM Apps

Assembly structure

My Asembly.dll
→ Assembly manifest (list)
→ Type metadata (details about class)

Type is another name for class
→ MSIL code
→ Resources

.cs file (C sharp)

.csproj → project file } only
.sln → solution file }

do not double click on .cs file
do not open folder

{} namespaces

functional overloading
optional parameter (int a, int b, int c=30)
optional arguments with default values

o.add(10,20,30) //positional parameters
o.add(c:30) //named parameters

Local functions
They can access variable declare outside.

19/7/22 Day 02

write method / non static member outside main method class

```
public class Class1
{ public int i;      field      class level variables
}                                by default 0.

{
    int i;      By default variable is private.
}
```

~~setter~~ & getter are not useful in expression in .Net then is property.

```
pirat int x;
public int X                    properties declon first letter capital
 {
     set
      {   // value is the rhs of o.x = 10;
          x = value;
      }
     get {
          return x;
         }
 }
```

Step into F11

1) To have validation in your data
2) future maintainer point of view.

Keep it as a property

auto property

```
public int P4 {get; set;}
```

dotNet say fields are fast but dont use instead use property.

IL disassembler
use to see what is inside assembler

1) Developer command propt vs 2019
2) ILDASM → EXIT

propertics → bin → debug → properties.exe

* Constructors & object initializers.

ctor

this → is referena to current object.

object initializer

class3 o2 = new class3() { x=10, P2=20, P3=30};

GC.Collect(); // slows down processing

static variable → singh (shared) copy for the cla

static function → can be called directly
classname.methodname
without creating object

in static methods you cannot access non static variables.

why static variables ? single copy
why properties ? validation
why static properties single copy validation.

why constructor → to initialize data member.
why static constructor → when the class is loaded

when is the static constructor called →
when the class is loaded
when is the class loaded →
when the first object is created & or
static members are called.

Static constructor is implicitly called &
implicitly private

static constructor → non static constructor will
called.

public static class Class1
{
only static members.
}

TODO
static class
can only have static members
connot create an object of static class

Inheritance

Inheritance → code reusability.

base / parent / super

derived / child / sub

→ single inheritance

```
class BaseClass{}

class derivedclass : Baseclass{}
```

Multiple inheritance → Not allowed in .Net

```
class baseclass1{}
class baseclass2{}

class derivedclass : baseclass1, baseclass2 {}
```

Multilevel inheritance

```
class Baseclass{}
class derivedclass : Baseclass{}
class subderivedclass : derivedclass{}
```

when we write a class by defaut it is inherited from object class.

Access specifiers

- private → same class
- public → everywhere
- protected → same class, derived class
- internal → same class, same assembly (same project)
- protected internal → same class, derived class same assembly (same project)
- private protected → same class, derived class which are in the same assembly

dll → did not have main class

Library

↓

class library (.Net framework)

Something defined within class then access specifiers by default is private

At namespace access specifier by default is internal

so write public class class1 { }

read only → use get & remove set

property accessor → private set { } get { }

or set { }

private get { }

either one can be private.

you can only reduce access, not increase it for property accessor.

20/7/2022

M T W T F S S
Page No.:
Date:
YOUVA

Base class members should pass to base class

Derivedclass(int i, int j) : base(i)

perived class members assign in derived class

Employee
GetNetSalary()
Manager
GetNetSalary()

1) Derived class can overload the base class method
same name, diff parameters.
Derived o = new Derived();
o.BaseMethod
o.DerivedMethod
Both methods are available from a derived class object.

2) Derived class can hide the base class method
Same same, same parameters.
* Any Method can be hidden
Derived o = new Derived();
o.DerivedMethod
only derived class method is available from a derived class object.

3) Derived class can override the base class method
Same name, same parameter

```
Derived o = new Derived();
o.DerivedMethod
```

* only a virtual method can be overridden

Virtual methods
Late bound - Run time binding - Run time polymorphism

base class method with keyword virtual
derived class method with keyword override

```
BaseClass o;
        o = new BaseClass();
        o.Display2();    non virtual early bound
        o.Display3();    virtual late bound

        o = new Derivedclass();
        o.Display2();
        o.Display3();    virtual method based on
                         memory
```

Oride method is also virtual method.

Sealed method is a method which cannot be override further → means you are stopping it

sealed override → is combination

late binding is slower than early binding
late binding is more flexible

In java all method by default virtual
In .Net all methods by default non virtual
early bond
& faster performance.

Abstract class

Employee()
clerk salesman officer

Employee o = new Employee(); ✗ you cannot instantiate but you can inherit abstract class

public abstract class c1
{

}

Abstract method → Pure virtual function is a virtual function which does not have method body only have method signature.

If a class has an abstract function then class must be abstract class
but All abstract class don't need to have abstract functions.

Todo

	Abstract class	Sealed class
Instantiate the class	No	Yes
Inherite the class	Yes	No

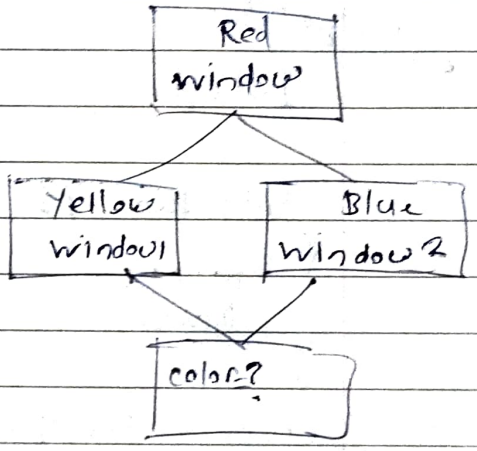
Sealed class is opposite of abstract class

→ Todo

Create sealed class

sealed class preventing code reusability
If you dont want to your code be reused then use sealed class.

Multiple inheritance



class Person
Name
Age >=0

class Employee : Person
Age >=18

class student : Person
Age >=2

class TrainerEmployee : Employee, student

Interface is like abstract class with all abstract methods
or only signature in it.

interface are not substitute for multiple inheritance

c# 7.3 version

All the methods in interface must implement in class → which is contract. both interface & class.

All methods in interface are by default public

todo → try inheritance in interfaces

How to write abstract property ?
employee abstract class

21/07/2022

Interface is like abstract class

.Net core feature

```
Interface I1
{
    M1();
    M2();
}

class C1 : I1
{
    M1(){}
    M2(){}
}
```

```
M3(){  // default implementation of interface method
}
```

This feature is supported in .Net core
Added in 8 version of c#

M3(){} // optional.

10.29 project started

through the interface call called default method but not by using class.C1.

using explicit implementation we can solve problem of ambuiguity

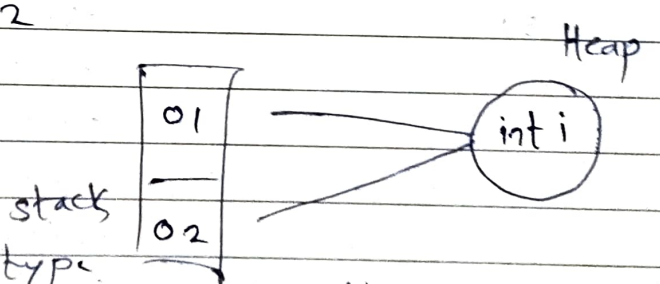
```
public void delete() { }
void I1.delete() { }
```

variables are not allowed in interface
properties are allowed in interface
but can have static members in interface

Reference & value types

```
Class1 o1 = new Class1();
Class1 o2 = new Class1();
```

o1 = o2



object/ref type
A ~~ref type~~ memory allocation is happening on Heap
& refere gets memory in stack

classes & variations:

Examples of reference types are classes, interfaces, delegates, arrays, the object class & string class are examples of reference type.

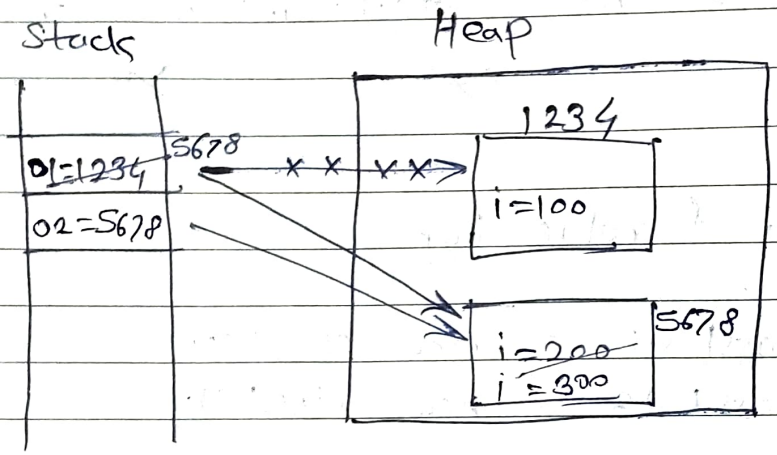
All delegates are classes
All arrays are also classes

What are the value types?

All structs & enums are value type
All value types stored in the stack

All enums internally stored as Integers
& all integers are structs
therefore all enums are structs

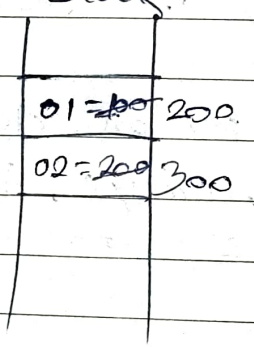
Ref types →




```
01 = 02
```

```
01.i = 100;
02.i = 200;
01 = 02;
02.i = 300;
cw(01.i); // 300
cw(02.i); // 300
```

Value types →

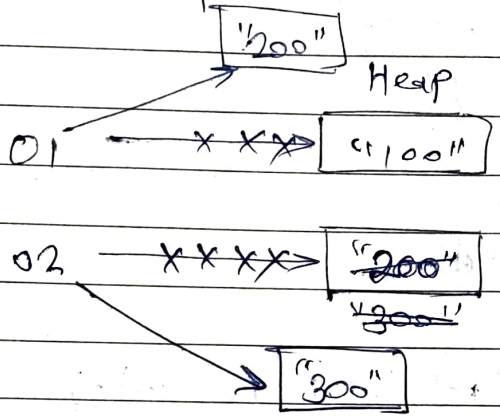



```
int 01, 02;
01 = 100;
02 = 200;
01 = 02;
02 = 300
cw(01); // 200
cw(02); // 300
```

string is only exception to all ~~classes~~ other reference types.

why they done this?

in c++ when pointer pointing to objects there is problem of dangling pointer




```
string o1, o2;
o1 = "100";
o2 = "200";
o1 = o2;
o2 = 300;
cw(o1); //200
cw(o2); //300
```

Data Types

Total 15 data types in C#

C# datatypes	cts data types		
int i;	// struct System.Int32	4 bytes	
uint ui;	// UInt32		
short s;	// Int16	2 bytes	
ushort us;	// UInt16		
long l;	// Int64	8 bytes	
ulong ul;	// UInt64		
byte b;	// Byte	1 bytes	value types
sbyte sb;	// sByte		
char ch;	// char	2 bytes	
bool bo;	// Boolean		
float f;	// single	4 bytes	
double dbl;	// Double	8 bytes	
decimal dec;	// Decimal	16 bytes → Highest precision	
string str;	// String		reference types
object o;	// object		reference types

important →

pass by ref / value

```
static void main()
{
    int a = 10        int c;
    int b = 20        int d;
    swap(ref a, ref b)
    cw (a); // 20
    cw (b); // 10
    init (out c, out d)
}

static void swap(ref int i, ref int j)
{
        int temp = i;
            i = j;
            j = temp;
}

static void init(out int i, out int j)
{
    i = 100;
    j = 200;
}
```

- When we use out it looses its initial value
out is similar to ref. changes made in
func reflect back in calling code
the initial value is discarded
out variable must be initialized in the func.

in is read only → you can't initialize

Struct — Faster than Heap operation

→ struct are value type → stored on stack
→ No Inheritance allowed in structs
→ Parameterless constructor not allowed in struct

enums

by defaul enums are integers

```
public enum TimeOfDay
{
    Morning,      by default 0    Morning = 100
    Afternoon,                1              101
    Evening;                  2              102
    Night                     3              103
}
```

public enum TimeOfDay : short
using enum it is easir to read, write the code

Nullabletypes — nullabletypes are value types

```
int? i;                 short? sh;
i = 10;                 decimal? dec;
i = null;

if (i.HasValue)
    j = i.value
else
    j = 0;

j = i.GetValueOrDefault();          default value = 0
j = i.GetValueOrDefault(10);        default value = 10
```

```
j = i ?? 10; //null coalescing operator
```

IDisposable

```
public class class1 : IDisposable
```

```
using (class1 o = new class1())
{
    o.Display();
}   // automatically calls IDisposable Dispose method.
```

22/07/2022

→ IDisposable

Arrays

dont' give name for namespace Array

```
int[] arr;
int[] arr = new int[5];
```

{0} → placeholder

```
arr[i] = int.Parse(console.ReadLine());
```

↓

return string

↓

int.parse

↓

convert string to int

```
arr[i] = convert.ToInt32(console.ReadLine());
```

int.Parse is faster than convert.ToInt32

String interpolation

```
$"The value of arr[{i}] is {arr[i]}"
```

foreach is used for displaying array element not for assigning values.

Array Class

```
Array.IndexOf(arr, 10)          → use linear search
                                  return -1 if not found.
Array.LastIndexOf(arr, 10)
Array.BinarySearch(arr, 10);
Array.Clear(arr, 0, arr.Length)
Array.Copy(arr, arr2, arr.Length)
Array.ConstrainedCopy(arr, 0, arr2, 0, arr.Length)
```

object array → int array → errors

```
Array.CreateInstance
Array.Reverse(arr);
Array.Sort(arr);
```

2-D Array

```
int[,] arr = new int[3,2]
```

3 Rows
2 Columns

```
arr.Length      → 6
arr.GetLength(0) → 3
arr.GetLength(1) → 2
arr.GetUpperBound(0) → 2   (3-1)
arr.GetUpperBound(1) → 1
GetLowerBound → always return 0.
```

arr.Rank → 2
↳ return dimensions