Day 1 : Working with C# using Visual Studio

Monday, October 11, 2021 10:12 AM

1.
• Overview of Writing Application by Using Visual C#
• Data Types, Operators, and Expressions
• Visual C# Programming Language Constructs
• Creating and Invoking Methods

- Overview of Writing Application by Using Visual C#
  What is the meaning/reason/Logic ?:

- A – **Assembly - Interacting Machines**
  Machine/hardware dependent
  Skillset that were Required was Very High
  Code was not Easy to follow - Load Register A9 = 20 LDA-A9-20, CLR ,
  Ex : IN **8085 MP** – 10-12 lined of Code just swapping to No
- **B – BASIC (Beginners All Purpose Symbolic instruction Code)**
  Johon Kemeny and Thomas Kurtz
  - FD 20, RT 30,
  - Bit English looking Syntax
  - Easier to Comprehend
  - Average Skills was not Demanding
  - Not machine Dependent - We were able to write in on machine and run on other machines
  - We were not able code in terms of real World Entities.
- **C – C (Dennis .Ritchie)**
  - POP – We were able to think in terms of Procedures
  - English Like (printf() and scanf())
  - Header files
  - Write Once and Run On any Machine – Platform Independent
  - Gave starting to High Level
  - Game Dev and Driver Implementation
  -

IBM their Own Version
C with Classes
C++
Apple - Objective C

150 Programming Lang :

Fortran
Pascal

.NET Platform : 15+ Programing
Wide Variety of Application
CLI – Console
GUI – Graphical
Web Application
Mobile Application | Xamrian |WPF|WCP
Game Development
Security Based Application

Of programming Lang

Anders heijlsberg -

C#.NET lang | VB.NET

Music Notions Sharp major #

C ++
++

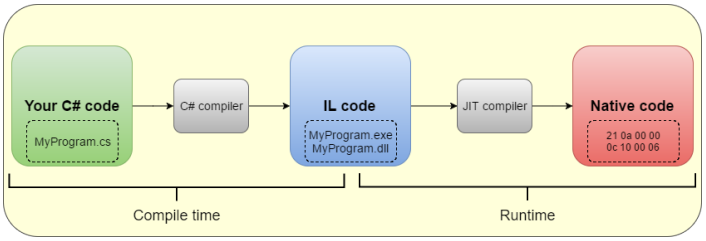
Why to create an Unifying lang

.NET is Moving towards OpenSource

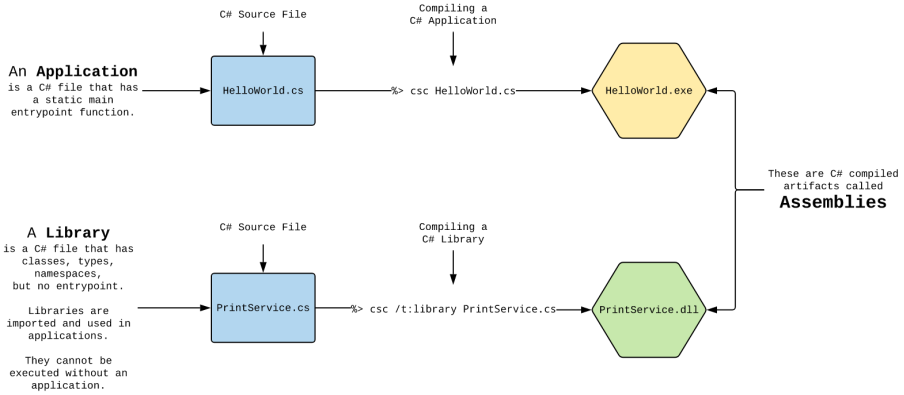
C# Features:

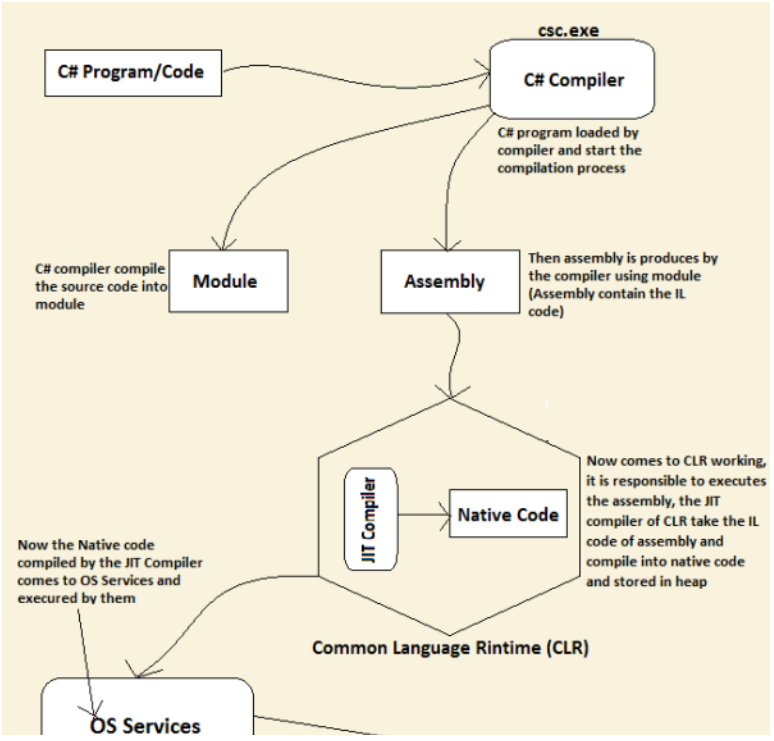
- It Follows Pascal Casing (Pascal A.H)
- Every File hass .CS extension
- First C# ode is Converted Into INternmediate COde (MSIL) - C# Complier - Roslyn
- MSIL code is converted into Native Code --- JIT Compiler
- Based OOP's (Object persistence _- Widely Supported in JAVA)
- **Console =! console**(Uppercase and Lowercase are different.
- Objectname.Classname

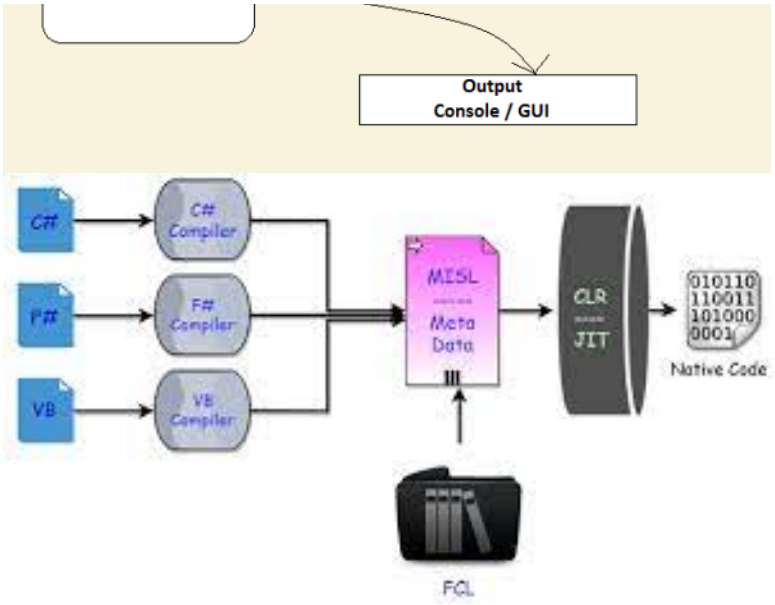
Raj.SalaryCalculation()



C# Compilation Process







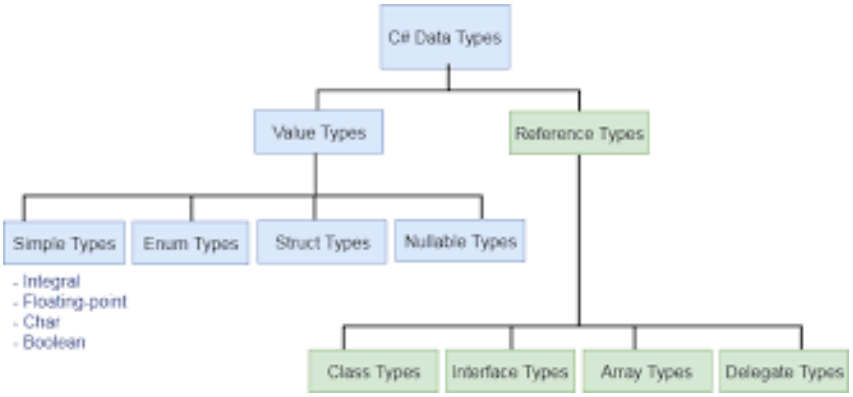
Store and retrieve data using literal and variable values in C#

Data Types in C#

Value Type – • Primitive (Integer, Floating point, Char, Boolean), Nullable Types User define - Enum Types, Struct Types,	**Reference Types-** • Class, Interface, Arrays & Delegates • Pre Define - Object and Strings

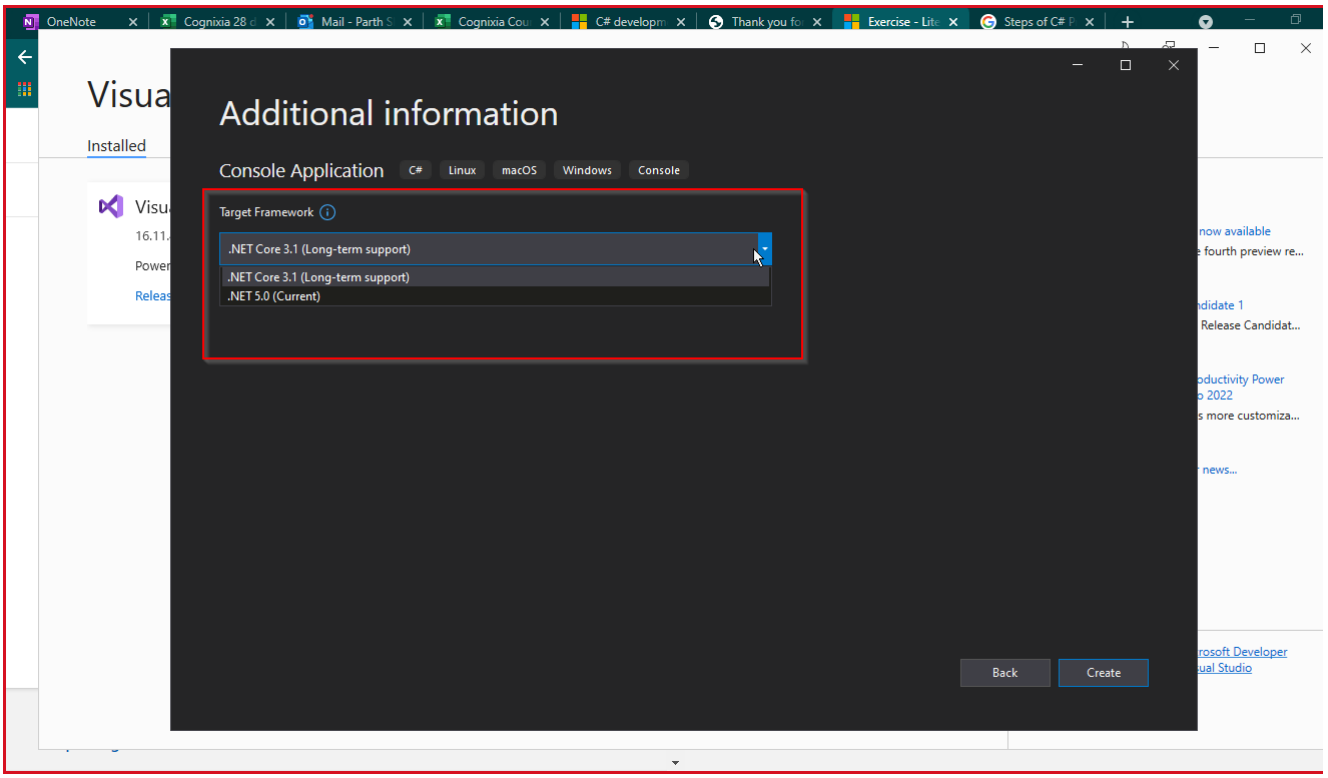
Int Age;
Stack implementation is used here
	Class Emp
	{
	}
	Heap memory Implementation

Faster Implementation	Bit Slower
Memory Overhead	Economical



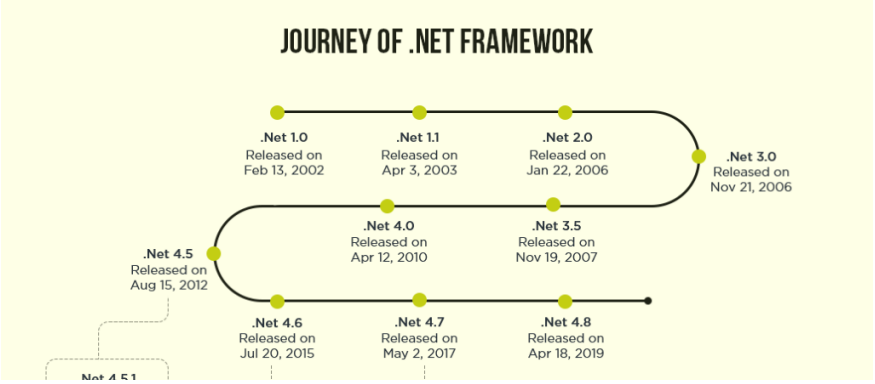
Data Types	Memory Size	Range

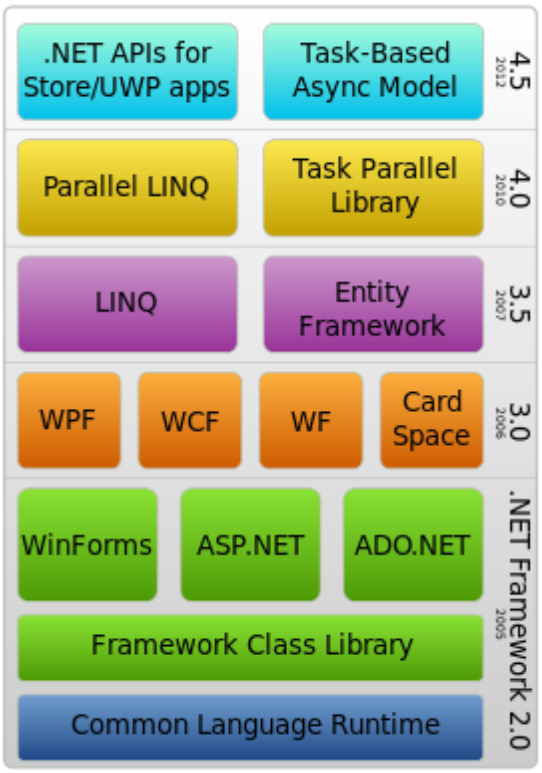
char	1 byte	-128 to 127
signed char	1 byte	-128 to 127
unsigned char	1 byte	0 to 255
short	2 byte	-32,768 to 32,767
signed Short	2 byte	-32,768 to 32,767
unsigned Short	2 byte	0 to 65,535
int	4 byte	-2,147,483,648 to 2,147,483,647
signed int	4 byte	-2,147,483,648 to 2,147,483,647
unsigned int	4 byte	0 to 4,294,967,295
long	8 byte	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
signed long	8 byte	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned long	8 byte	0 to 18,446,744,073709,551,615
float	4 byte	$1.5 * 10^{-45}$ to $3.4 * 10^{38}$ (7 Digit)
double	8 byte	$5 * 10^{-324}$ to $1.7 * 10^{308}$
decimal	16 byte	$-7.9 * 10^{28}$ to $7.9 * 10^{28}$

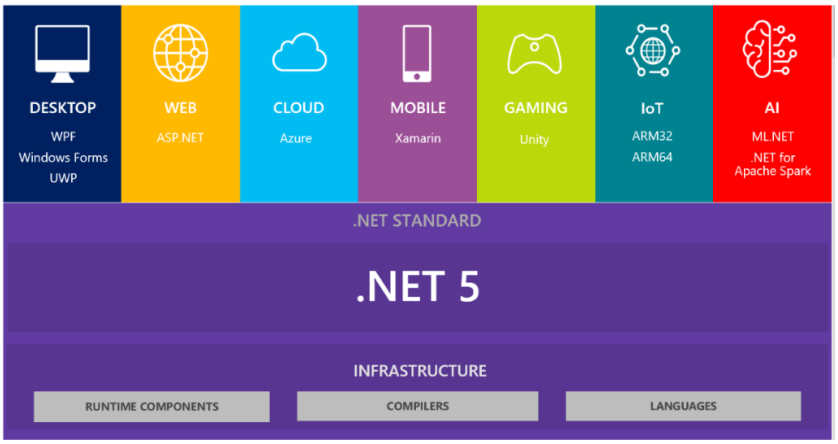


.NET Framework:

Virtual Structure







Typecasting : Boxing/Unboxing

Int Age = 20;

Distance from Dun to moon = 2.45 *10^28

Value Type to Reference =
Reference type to value Type =

We take help of Function Overloading

Speak()

Speak(Human) - Talking
Speak(Dog) ---- Barking
Speak(Duck) --- Quack
Speak(Birds)---- Chirping

Speak() - Can be called in 4 different ways according to the scenario.
This is known as Function Overloading

Ex Convert.toString() - 36 Different ways to take an input

Implicitly Typed variable :

- var name = "raj";
- var Age = 21;
- var Qualified = true;

Password, Keys, Adress

Operator Overloading :

20 + 5 = 25 // Addition
Parth + Shukla = ParthShukla // Concatenation

Collections :
Generic Type of Collections : Array
Non Generic types pf collection:

Commonly used collection types

Collection types represent different ways to collect data, such as hash tables, queues, stacks, bags, dictionaries, and lists.

All collections are based on the ICollection or ICollection<T> interfaces, either directly or indirectly. IList and IDictionary and their generic counterparts all derive from these two interfaces.

In collections based on IList or directly on ICollection, every element contains only a value. These types include:

- Array
- ArrayList
- List<T>
- Queue
- ConcurrentQueue<T>
- Stack
- ConcurrentStack<T>
- LinkedList<T>

In collections based on the IDictionary interface, every element contains both a key and a value. These types include:

- Hashtable
- SortedList
- SortedList<TKey,TValue>
- Dictionary<TKey,TValue>
- ConcurrentDictionary<TKey,TValue>

The KeyedCollection<TKey,TItem> class is unique because it is a list of values with keys embedded within the values. As a result, it behaves both like a list and like a dictionary.

When you need efficient multi-threaded collection access, use the generic collections in the System.Collections.Concurrent namespace.

The Queue and Queue<T> classes provide first-in-first-out lists. The Stack and Stack<T> classes provide last-in-first-out lists.

Strong typing

Generic collections are the best solution to strong typing. For example, adding an element of any type other than an Int32 to a `List<Int32>` collection causes a compile-time error. However, if your language does not support generics, the System.Collections namespace includes abstract base classes that you can extend to create collection classes that are strongly typed. These base classes include:

- CollectionBase
- ReadOnlyCollectionBase
- DictionaryBase

How collections vary

Collections vary in how they store, sort, and compare elements, and how they perform searches.

The SortedList class and the SortedList<TKey,TValue> generic class provide sorted versions of the Hashtable class and the Dictionary<TKey,TValue> generic class.

All collections use zero-based indexes except Array, which allows arrays that are not zero-based.

You can access the elements of a SortedList or a KeyedCollection<TKey,TItem> by either the key or the element's index. You can only access the elements of a Hashtable or a Dictionary<TKey,TValue> by the element's key.

Use LINQ with collection types

The LINQ to Objects feature provides a common pattern for accessing in-memory objects of any type that implements IEnumerable or IEnumerable<T>. LINQ queries have several benefits over standard constructs like `foreach` loops:

- They are concise and easier to understand.
- They can filter, order, and group data.
- They can improve performance.

For more information, see LINQ to Objects (C#), LINQ to Objects (Visual Basic), and Parallel LINQ (PLINQ).

Title	Description
Collections and Data Structures	Discusses the various collection types available in .NET, including stacks, queues, lists, arrays, and dictionaries.
Hashtable and Dictionary Collection Types	Describes the features of generic and nongeneric hash-based dictionary types.
Sorted Collection Types	Describes classes that provide sorting functionality for lists and sets.
Generics	Describes the generics feature, including the generic collections, delegates, and interfaces provided by .NET. Provides links to feature documentation for C#, Visual Basic, and Visual C++, and to supporting technologies such as reflection.

Write a C# Application for checking If a user is allowed to vote or not ??

Step 1 : Ask for name and Age of the user
Step 2 : Check if age < 18 then Not allowed for voting
Step 3 : You not allowed for voting

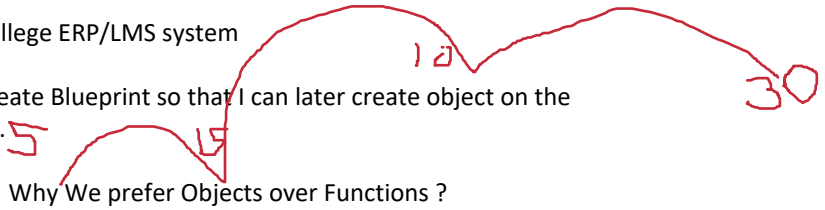
```
if (Age>= 18)
        {
           Console.WriteLine("You can vote");
        }
        else
        {
           Console.WriteLine("You can't vote");


        }
```

Class and Object :

- Class is Blue Print for an Object
- Class is template for creating instances of Object

College ERP/LMS system

Create Blueprint so that I can later create object on the go.



Why We prefer Objects over Functions ?
POP – It is completely based on procedures

OOP - is based on classes and Object

- Kid : --- wakeUp()
  GotoSchool()
  Playing()
  SummerVacations()

- College Student -
  Wakeup( )
  Collegelecures()
  BunkLectures()
  WatchingMovies()
  SummerBreaks()

- Professional-
  Wakeup()
  Meetings/Calls()
  Leaves()
  NoSummerBreak()

Functionality keeps on Changing with Changing times

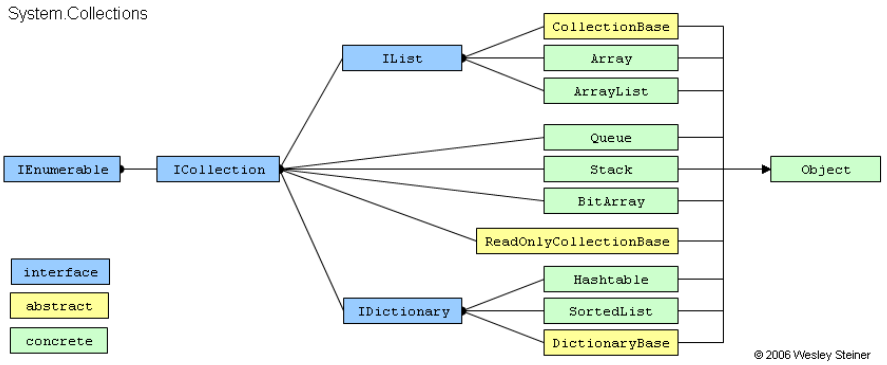
Objects remains same

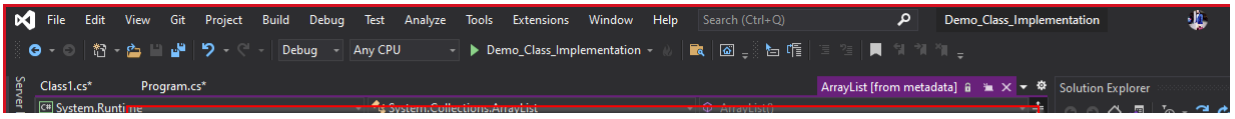
```
LMS : Class
{
    Student_Name,
    Email,
    MobileNo
    Assessment Day,
    AsessmentTopic,
    Qualify()
}

    LMS Raj = new LMS()
    //Creating instance of the class
```

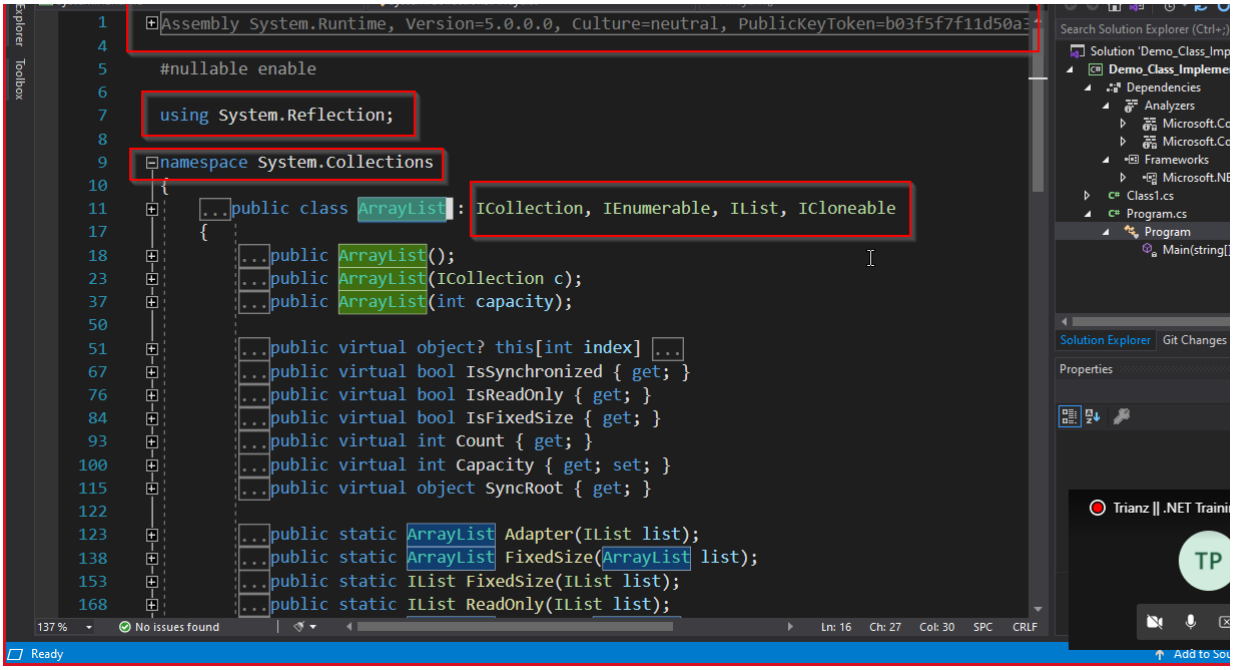
Que: What should we do when we want to Add More Functionality to the previously defined class, Without altering it ??
Ans: We define both classes as **partial**

Parial Design team Classes
Partial development Team Classes

They get combined into one class at run time.







Classes are inherited
Interfaces are implemented
Concrete Classes

Interfaces in C#

- We can define/declare methods

While Creating a Car :

- Few Functionalities are predefine : Circuit boards are imported
- Few functionalities are added as new features: Chassis is manufactured
- Few Functionalities are removed/replaced : Acessories

All About Interfaces in C#:

- An interface can be a member of a namespace or a class. An interface declaration can contain declarations (signatures without any implementation) of the following members:
- Methods
- Properties
- Indexers
- Events
- An interface can inherit from one or more base interfaces. When an interface overrides a method implemented in a base interface, it must use the explicit interface implementation syntax.

  When a base type list contains a base class and interfaces, the base class must come first in the list.

  A class that implements an interface can explicitly implement members of that interface. An explicitly implemented member cannot be accessed through a class instance, but only through an instance of the interface. In addition, default interface members can only be accessed through an instance of the interface.

Assemblies are the smallest unit of Code in .NET

Task : Implement All types of Classes and Interface using the Concept of Real World Entities (Feedback Management system.