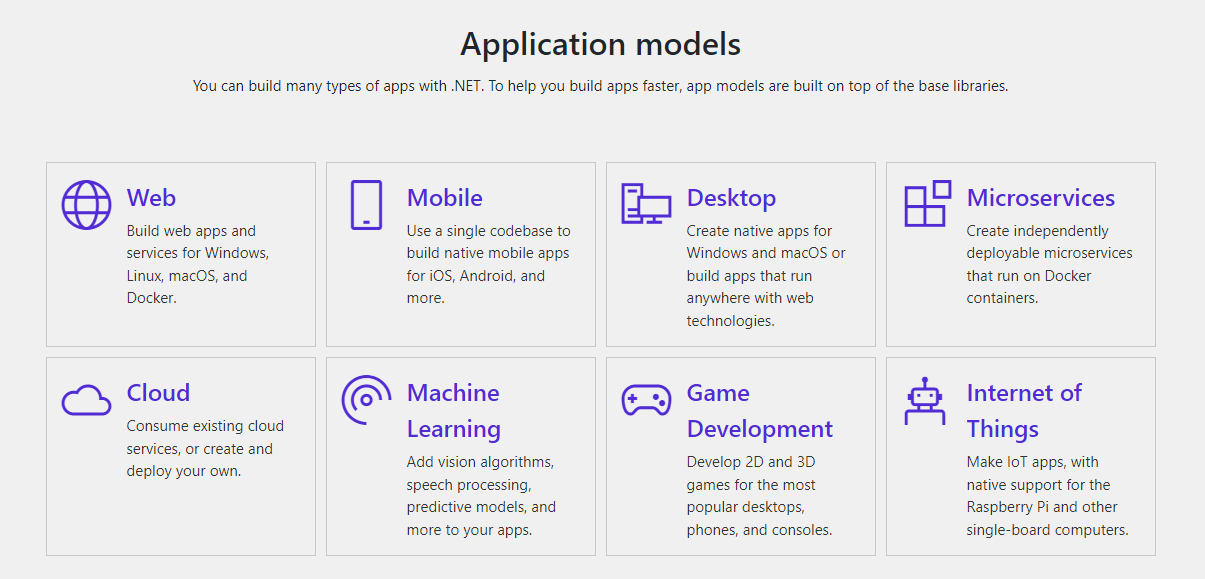
Introduction to 4.6 .NET framework

1.What is .NET Platform?

Dot Net(.Net) is a free, cross-platform, open source developer platform for building many different types of applications.

With .NET, you can use multiple languages, and libraries to build for web, mobile, desktop, games, Ios, and more.



2 What is .NET Framework ?

.Net made up of tools, programming languages, and libraries for building many different types of applications.

There are various implementations of .NET. Each implementation allows .NET code to execute in different places—Linux, macOS, Windows, iOS, Android, and many more.

* .NET Framework is the original implementation of .NET. It supports running websites, services, desktop apps, and more only on Windows.

1. .NET is a cross-platform implementation for running websites, services, and console apps on Windows, Linux, and macOS.

.NET is open source on GitHub. .NET was previously called .NET Core.

1. .Xamarin/Mono is a .NET implementation for running apps on all the major mobile operating systems, including iOS and Android.
2. .NET Standard is a formal specification of the APIs that are common across .NET implementations. This allows the same code and libraries to run on different implementations.

.NET Framework applications are written in C#, F#, or Visual Basic and compiled to Common Intermediate Language (CIL). The Common Language Runtime (CLR) runs .NET applications on a given machine, converting the CIL to machine code.

3 What is the difference between .NET and .NET Framework?

* .NET and .NET Framework share many of the same components and you can share code across the two. Some key differences include:
* .NET is cross-platform and runs on Linux, mac OS, and Windows. and .NET Framework only runs on Windows.
* .NET is open-source and accepts contributions from the community. The .NET Framework source code is available but doesn't take direct contributions.
* All of the innovation happens in .NET.
* .NET Framework is included in Windows and automatically updated machine-wide by Windows Update. .NET is shipped independently.

4 Dot Net(.Net) Framework, Languages, and Tools

Languages

You can write .NET apps in C#, F#, or Visual Basic.

C# is a simple, modern, object-oriented, and type-safe programming language.

F# is a programming language that makes it easy to write succinct, robust, and performant code.

Visual Basic is an approachable language with a simple syntax for building type-safe, object- oriented apps.

Tools

The Visual Studio product family provides a great .NET development experience on Windows, Linux, and mac OS.

The Visual Studio Marketplace has thousands of editor extensions from Microsoft and others.

If you prefer to use a different editor, there are .NET command-line tools and plug-ins for many popular editors.

5 Dot Net(.Net) Framework Major Components

The two major components of .NET Framework are

* The Common Language Runtime (CLR).

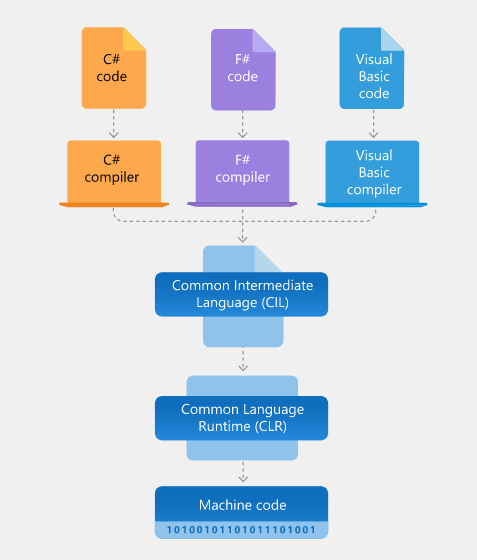
The Common Language Runtime (CLR) is the execution engine that handles running applications. It provides services like thread management, garbage collection, type-safety, exception handling, and more.

* The .NET Framework Class Library.

The Class Library provides a set of APIs and types for common functionality. It provides types for strings, dates, numbers, etc. The Class Library includes APIs for reading and writing files, connecting to databases, drawing, and more.

.NET applications are written in the C#, F#, or Visual Basic programming language. Code is compiled into a language-again Common Intermediate Language (CIL). Compiled code is stored in assemblies—files with a .dll or .exe file extension.

When an app runs, the CLR takes the assembly and uses a just-in-time compiler (JIT) to turn it into machine code that can execute on the specific architecture of the computer it is running on.



5 Common Language Runtime (CLR)

6 Understand the .Net Framework .

7 Introduction to .Net Core

Introduction to C#

1.Features of C#.

Strongly typed: It means in C# whenever we going to use r store any data we should specify the datatype its data like String r number

Objected oriented: oops Component oriented: it specifies us to separate each task that means if u developing app u must undergo many task where in that means each task must be defined in different block so that it is independent of each other and not dependent of another.

Event driven: It is nothing but which like runtime program Example: A calculator here what happens calculator works completel on runtime event like whether it is adding r subtracting r div r multi it its taking numbers at that particular time & performing its event driven

Task driven: it is something like performing asynchronous programing means if something high priority task it will execute that first and remaining later.

MAIN FEATURES OF C#

a) SIMPLE

Pointers are missing in C#.

Unsafe operations such as direct memory manipulation are not allowed.

In C# there is no usage of "::" or "->" operators.

Since it's on .NET, it inherits the features of automatic memory management and garbage collection.

Varying ranges of the primitive types like Integer, Floats, etc.

Integer values of 0 and 1 are no longer accepted as boolean values. Boolean values are pure true or false values in C# so no more errors of "="operator and "=="operator.

"==" is used for comparison operation and "=" is used for assignment operation.

b) MODERN

C# has been based according to the current trend and is very powerful and simple for building interoperable, scalable, robust applications.

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.

c) OBJECT ORIENTED

C# supports Data Encapsulation, inheritance, polymorphism, interfaces.

(int, float, double) are not objects in java but C# has introduces structures(structs) that enable the primitive types to become objects.

int i=1; string a=i Tostring(); //conversion (or) Boxing

d) TYPE SAFE

In C# we cannot perform unsafe casts like convert double to a boolean.

Value types (primitive types) are initialized to zeros and reference types (objects and classes) are initialized to null by the compiler automatically.

arrays are zero base indexed and are bound checked.

e) INTEROPERABILITY

C# includes native support for the COM and windows based applications.

Allowing restricted use of native pointers.

Users no longer have to explicitly implement the unknown and other COM interfacers, those features are built-in.

C# allows the users to use pointers as unsafe code blocks to manipulate your old code.

Components from VB NET and other managed code languages and directly be used in C#.

f) SCALABLE AND UPDATEABLE

.NET has introduced assemblies that are self-describing by means of their manifest. manifest establishes the assembly identity, version, culture and digital signature etc. Assemblies need not to be registered anywhere.

To scale our application we delete the old files and updating them with new ones. No registering of dynamic linking library.

Updating software components is an error prone task. Revisions made to the code. can effect the existing program C# support versioning in the language. Native support for interfaces and method overriding enable complex frameworks to be developed and evolved over time.

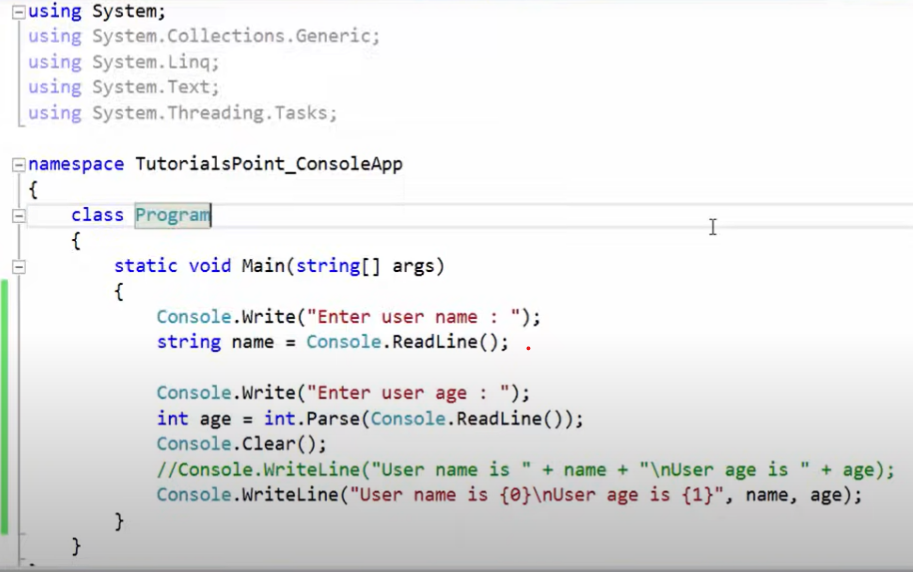
CONCLUSION

C# is a modern, type safe programming language, object oriented language that enables programmers to quickly and easily build solutions for the Microsoft .NET platform.

2. Overview of Visual Studio.

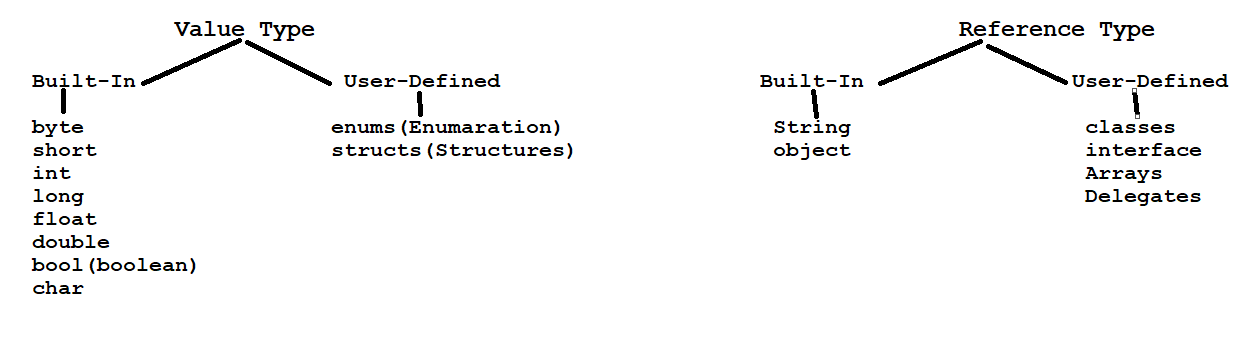
3. General Structure of a C# Program.

4. C# Compilation and Execution. Creating and Using a DLL

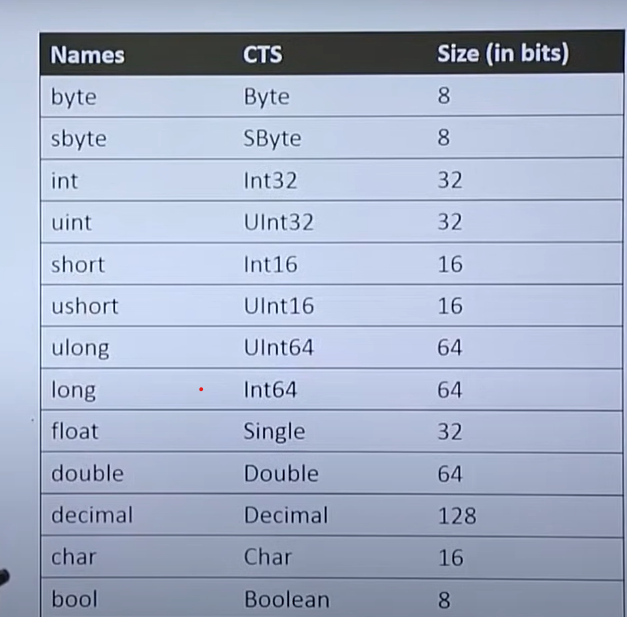


Fundamentals of C#

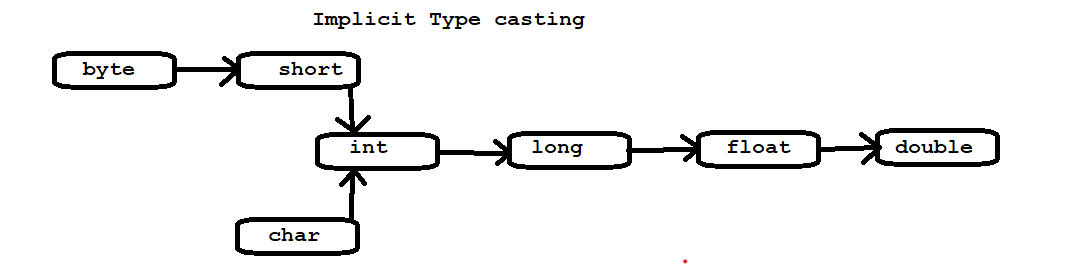
1.Data Types in C#? Value Types and Reference Types

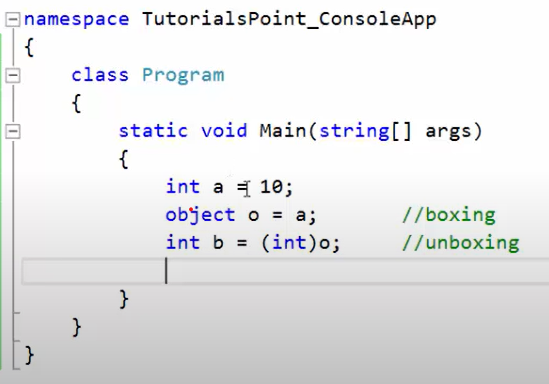


Sizes of inbuilt types

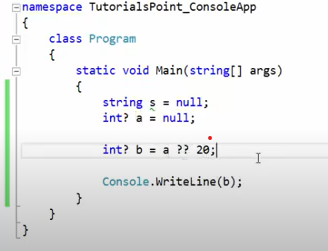


2. Boxing and Unboxing? Implicitly Typed Local variables?





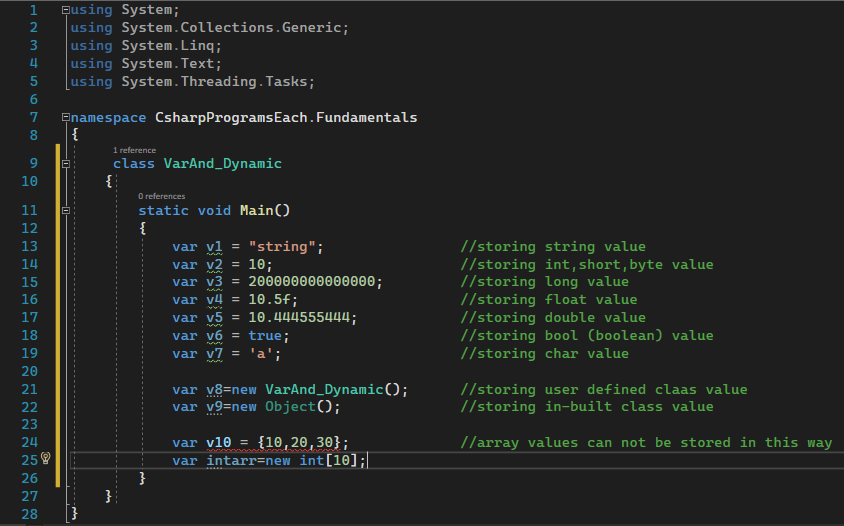
3.Nullable Types.



4.Var and Dynamic.

Var and Dynamic is also kind of datatype but these both can hold all in-built user-defined data’s .

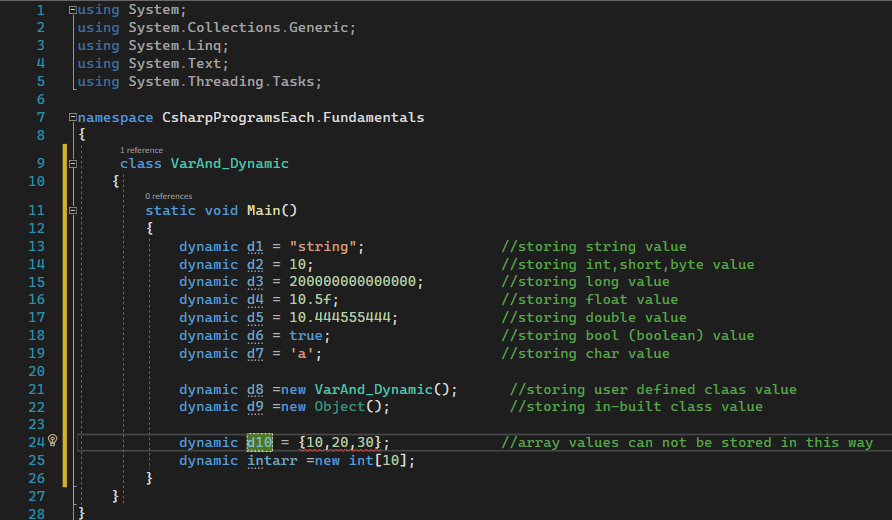
Var



Do not use var to declare the data type for the property (properties) as well as the return type of the method (method) in the class.

Dynamic

The dynamic keyword is used to declare dynamic types. The dynamic types tell the compiler that the object is defined as dynamic and skip type-checking at compiler time; delay type-checking until runtime. All syntaxes are checked and errors are thrown at runtime.



Operators

Arithmetic operators : + , - , / , \*, %

Relational operators : < , > , <= , >= ,== , !=

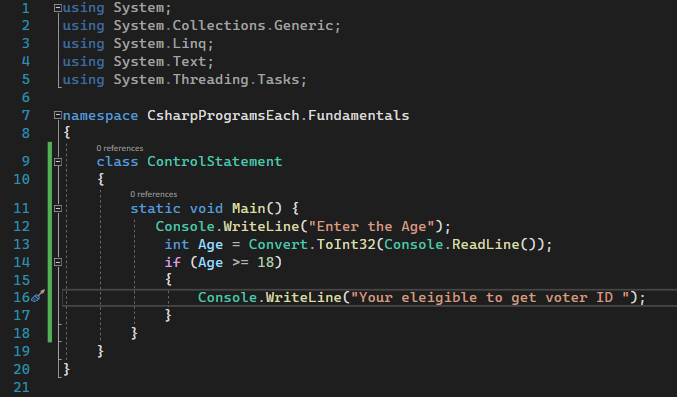
Logical Operator : AND (&&), OR (||), NOT (!)

Assignment operator: = ,+= ,-=, \*=, /=, %=

Special operator : + , increment++ , decrement- -

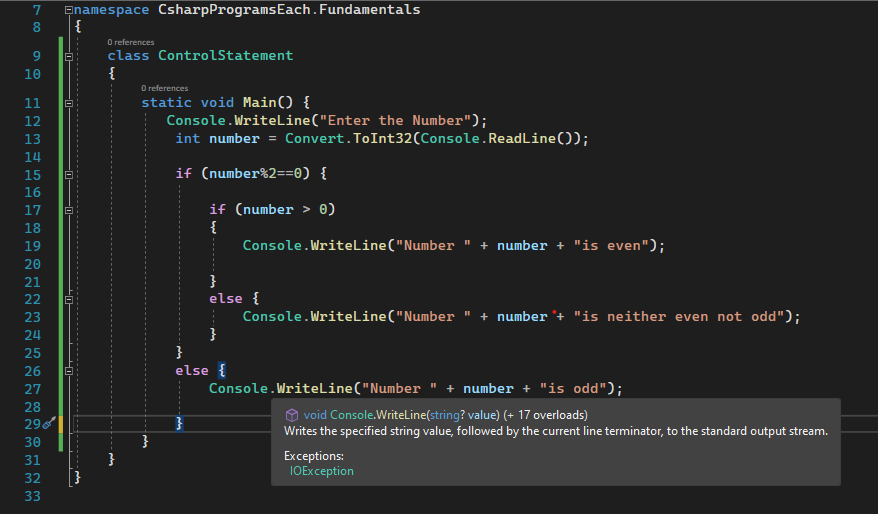
Flow Control Statement

If Statement

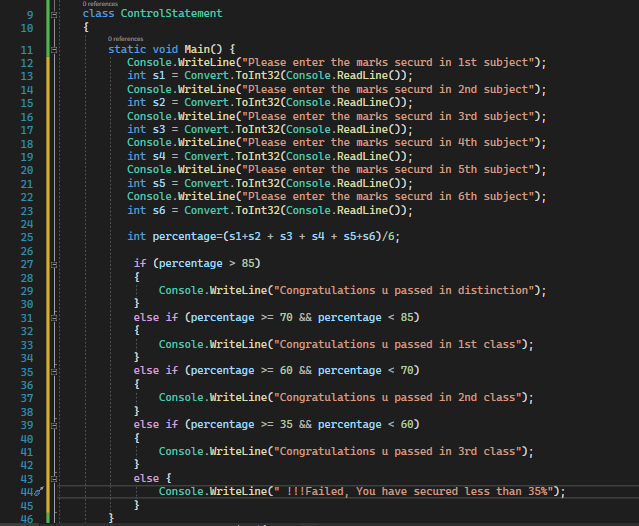


If and Else

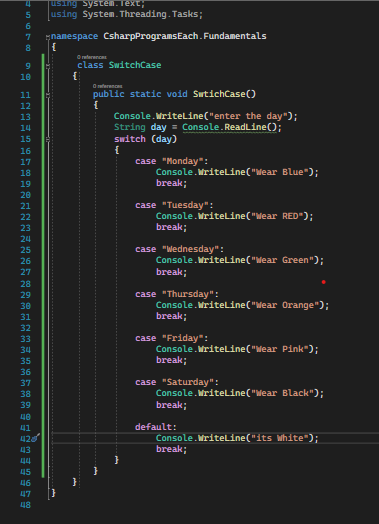
Nested If



Ladder if and else

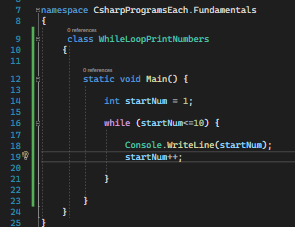


Switch case

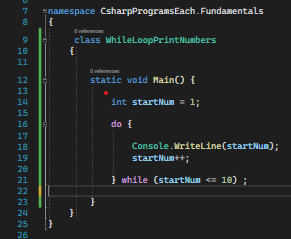


Looping statements

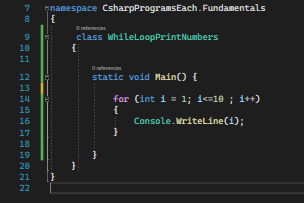
While Loop



Do while



For Loop



5.The ‘OBJECT’ Base class (super most class) in .Net?

Object class contains basically 2 constructor and 7 function member they are:

Constructors

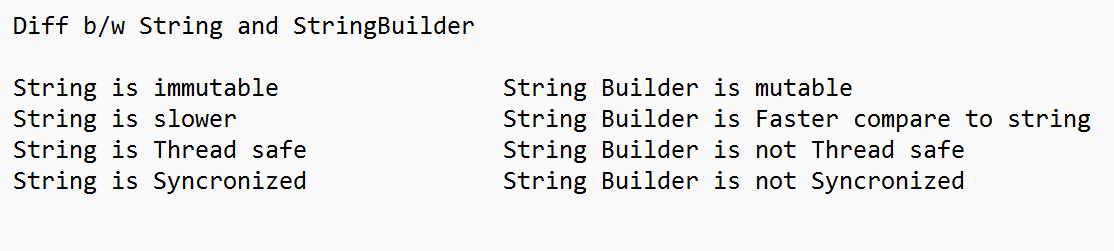
* + Object() : Initializes a new instance of the System.Object class.
  + ~Object() : Allows an object to try to free resources and perform other cleanup operations

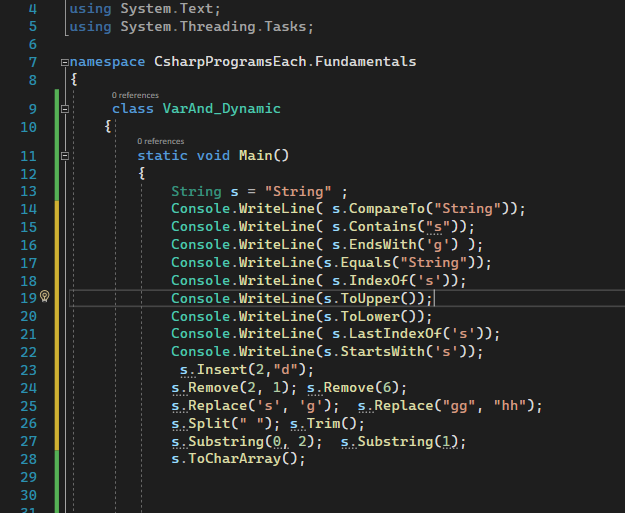
Function Memebers

* 1. Equals(object? obj)
  2. Equals(object? objA, object? objB)
  3. GetHashCode()
  4. MemberwiseClone()
  5. ReferenceEquals(object? objA, object? objB)
  6. ToString()
  7. GetType() : Gets the System.Type of the current instance.

6. String and String class Methods.

* Compare() : Compare two strings and returns integer value as output. It returns 0 for true and 1 for false.
* Contains() : The C# Contains method checks whether specified character or string is exists or not in the string value.
* EndsWith() : This EndsWith Method checks whether specified character is the last character of string or not.
* Equals() : The Equals Method in C# compares two string and returns Boolean value as output.
* IndexOf() : Returns the index position of first occurrence of specified character.
* ToLower() : Converts String into lower case based on rules of the current culture.
* ToUpper() : Converts String into Upper case based on rules of the current culture.
* Insert() : Insert the string or character in the string at the specified position.
* LastIndexOf() : Returns the index position of last occurrence of specified character.
* Remove() : This method deletes all the characters from beginning to specified index position.
* Replace() : This method replaces the character.
* Split() : This method splits the string based on specified value.
* StartsWith() : It checks whether the first character of string is same as specified character.
* SubString() : This is a overloaded method used to retrieve value from specified position.
* ToCharArray() : Converts string into char array.
* Trim() : It removes extra whitespaces from beginning and ending of string.
* Length- : It is a string property that returns length of string.

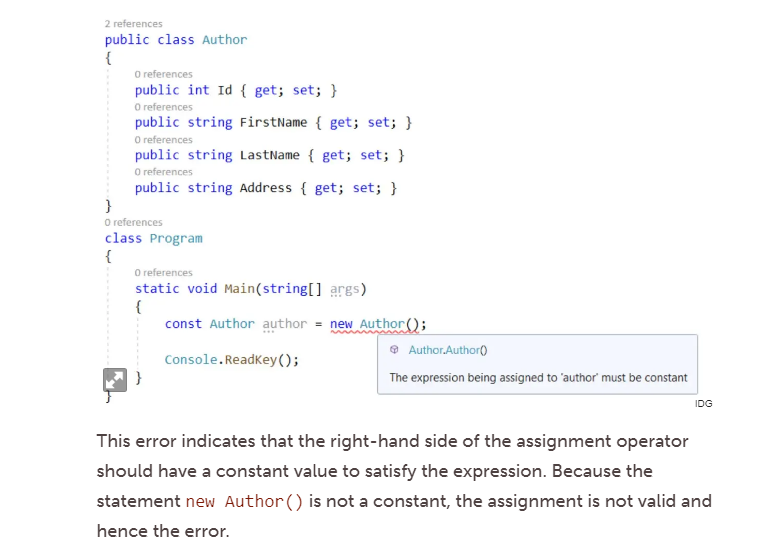




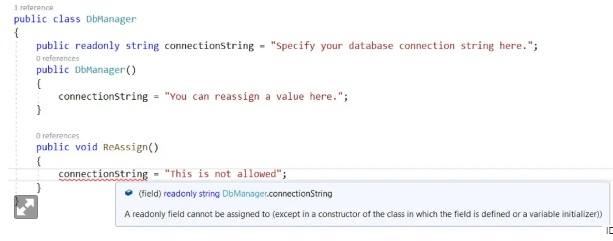
14 Parse() vs TryParse() vs Convert Class methods

15.Const, Readonly.

The const (read: constant) keyword in C# is used to define a constant variable, i.e., a variable whose value will not change during the lifetime of the program. Hence it is imperative that you assign a value to a constant variable at the time of its declaration.



The readonly keyword can be used to define a variable or an object as readable only. This means that the variable or object can be assigned a value at the class scope or in a constructor only. You cannot change the value or reassign a value to a readonly variable or object in any other method except the constructor.



Here is the rule of the thumb you can follow when working with the const, readonly, and static keywords. Use the const keyword when the value contained in a variable will never change during the lifetime of the application. Use the readonly keyword when you are not sure whether the value of a variable of an object needs to change but you want to prevent other classes from changing the value. Use the static keyword when you want the member of a class to belong to the type rather than to the instance of the type.

Oops with C#

1.Access modifiers.

In C# we have 5 modifiers They are:

1.Public. 2. Private. 3. Protected. 4. Internal. 5. Protected internal.

Private: private modifier whenever we declare any properties and behaviours pre-fixed with private keyword that means we can access only within same class.

* Within the same class.

Public: public modifier if the properties and behaviours is pre-fixed with public keyword means that properties and behaviours can be accessed.

* Within the same class.
* Different class with inheritance and without inheritance.
* Within same project any class.
* Same solution different project with reference added.

Protected: protected modifier if the properties and behaviours is pre-fixed with protected keyword means that properties and behaviours can be accessed.

* Within the same class.
* Different class with inheritance.
* Same solution different project with reference added and with inheritance.

Internal: internal modifier if the properties and behaviours is pre-fixed with internal keyword means that properties and behaviours can be accessed.

* Within the same project any class.

Protected Internal: protected internal modifier if the properties and behaviours is pre-fixed with protected internal keyword means that properties and behaviours can be accessed.

* Within the same project any class.
* Same solution different project with reference added and with inheritance.

Constructors

Constructor is a special member of a class where in its name should be same as class name.

Syntax

<Access Modifiers> <Access specifiers> ClassName ()

{

}

Points on constructors

* Constructor is a special member cause if programmer forgets or missess to create a constructor compiler will create a default constructor.
* Constructor doesn’t return anything. (doesn't have return type).
* Constructor can be overloaded.
* Constructor is a main reason for inheritance without constructor inheritance cannot be achieved.
* It is used to create object.
* Constructor is used for initializing instance variables.
* Static constructor is used to initialize static variables.
* Constructor can be called in three ways they are:

1. this ()
2. base ()
3. new Keyword

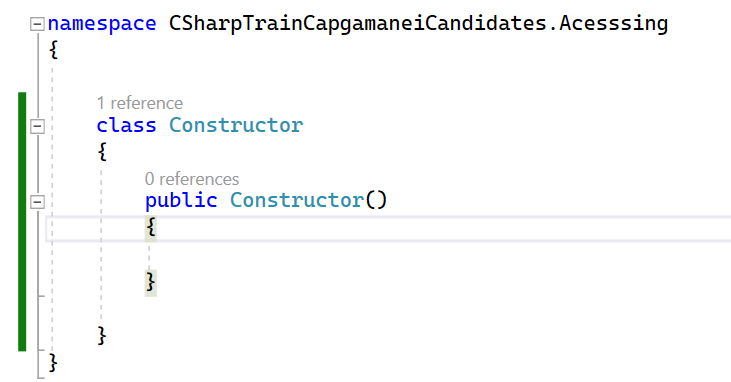
Types of Constructor

* Default constructor.
* Parameterized constructor/User-Defined constructor.
* Copy constructor.
* Static constructor.

Default Constructor:

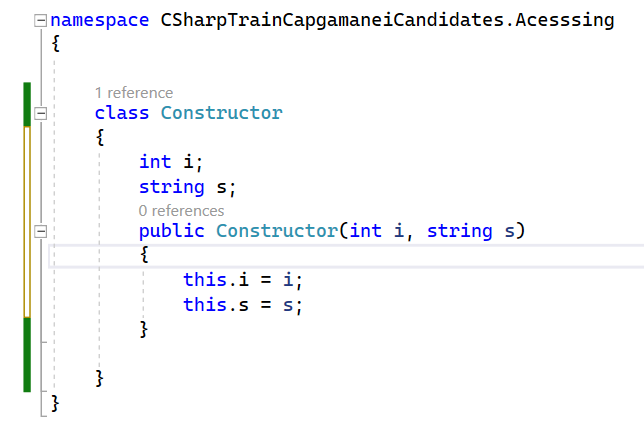
Default constructor is nothing but Constructor which is generated by compiler whenever programmer forgets to create a constructor inside a class.

* Default constructor is always non-parameterized.
* Constructor is used to initialize data members of class.



Parameterized constructor/User-Defined constructor:

Whenever the constructor is defined by programmer explicitly inside a class it is known as user-defined constructor, this user defined constructor can be parameterized or non-parameterized constructor.



Copy constructor:

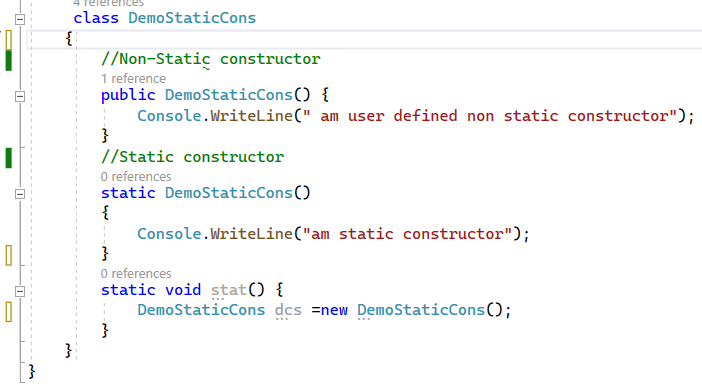
Copy constructor are nothing but if we want to create multiple instances with same values then we use these copy constructors, here these copy constructor takes its own class constructor as parameter it is known as copy constructor.

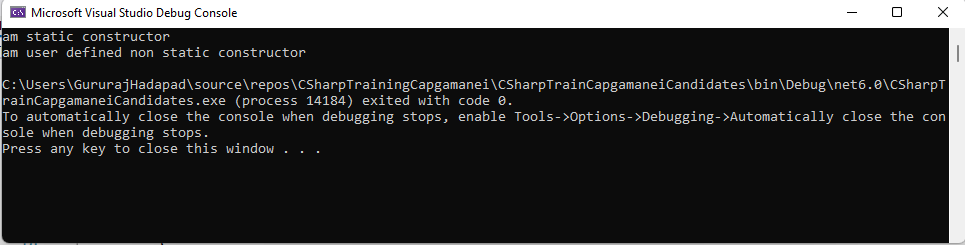


Static constructors:

Static constructor is a constructor when programmer declares a constructor explicitly with static keyword we call it as static constructor.

* If a class contains static variables only then implicit static constructor will be present inside a class or else, programmer need to declare it explicitly.
* Static constructors are responsible to initialize static variables.
* These constructors are never called explicitly.
* These constructors are first to execute under any class main method.
* Since static constructors are implicitly called cannot be called explicitly these constructors cannot be overloaded.



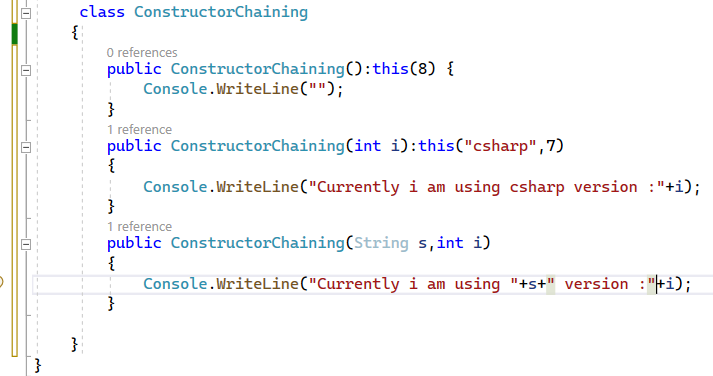


Constructor Chaining:

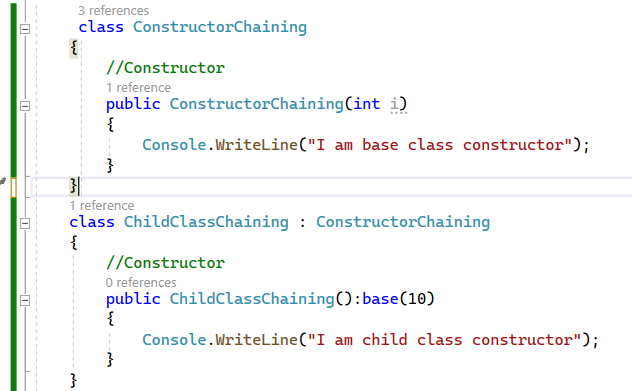
Constructor chaining is nothing but calling one constructor inside another constructor it is known as constructor chaining.

Constructor cannot be called using its name. So constructor should be called with this() and base().

this () calling statement: this () calling statement is used to call the one constructor inside another constructor of same class.



base () calling statement: base () calling statement is used to call the one constructor inside another constructor of its base class (parent class).



Inheritance

What is inheritance.

Accessing properties from one class to another class is known as Inheritance.

Inheriting properties from one class to another class is known as Inheritance.

Inheriting properties from Super class to Sub class is known as Inheritance.

Inheritance is achieved by using ' : ' Symbol in CSharp.

Advantages of inheritance.

To avoid code redundancy(duplication).

To reuse code.

To achieve polymorphism.

Note:

We can inherit only properties and behaviours.

We cannot inherit constructors and private members of super class.

Using sub class reference we can use parent class members.

Using Parent class reference we cannot use sub class members.

Using inheritance, we can achieve generalization.

Types of inheritance

Single Level inheritance.

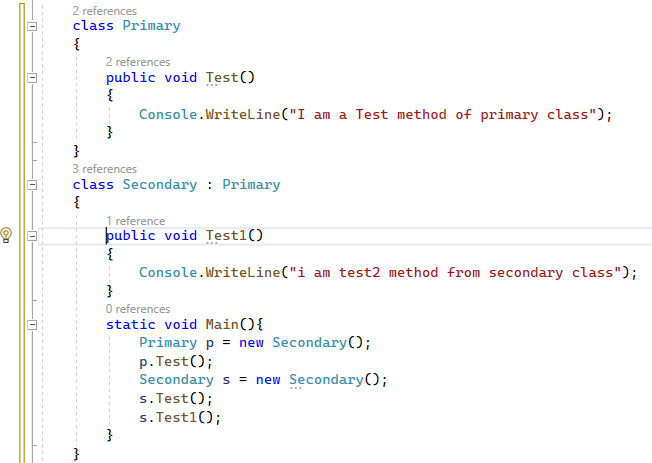
Multi-level inheritance.

Hierarchical inheritance

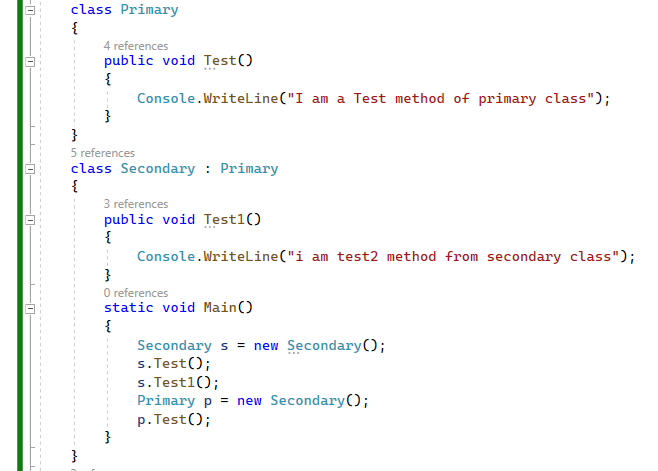
Multiple Level inheritance.

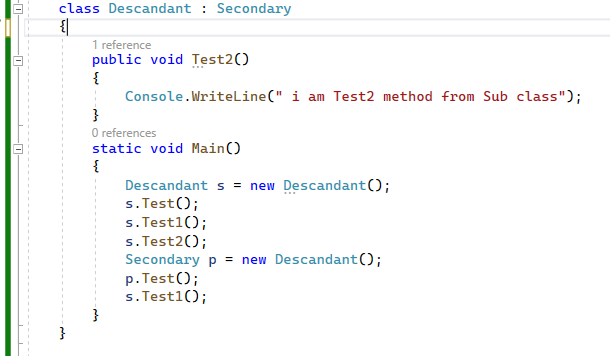
Hybrid Inheritance.

Single Level inheritance.

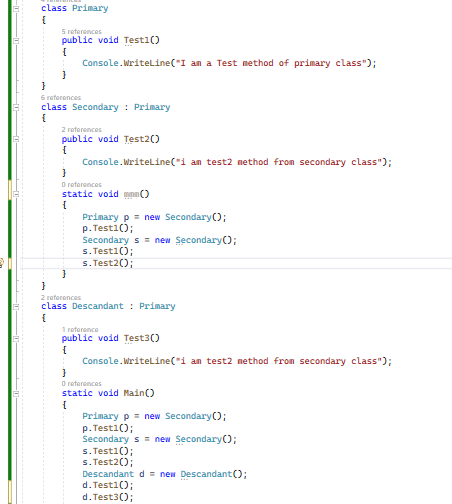


Multi-level inheritance.





Hierarchical inheritance:



Methods

Methods are nothing but behaviour of an instance or multiple actions executed under one block

Syntax

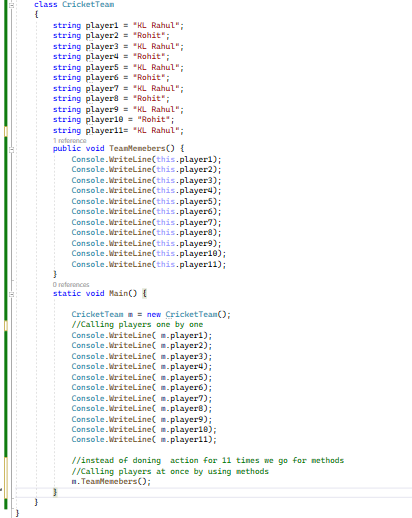
<Access modifier> <Access specifier> <Return Type> Method Name(arguments/parameter){

<Method Body>

<Method implementation>

<Method definition>

}



Points on methods.

* Access modifiers, Access specifiers and parameters/arguments are optional but return type and method name are mandatory.
* We have to create method inside the class outside the main method.
* Inside a class we can create n number of methods.
* Inside one method we call another method.
* We can also call inside method same method
* We can call method by its method signature.

Method Duplication.

Inside a class when we have multiple methods with same but not satisfying method overloading conditions it is known as method duplication.

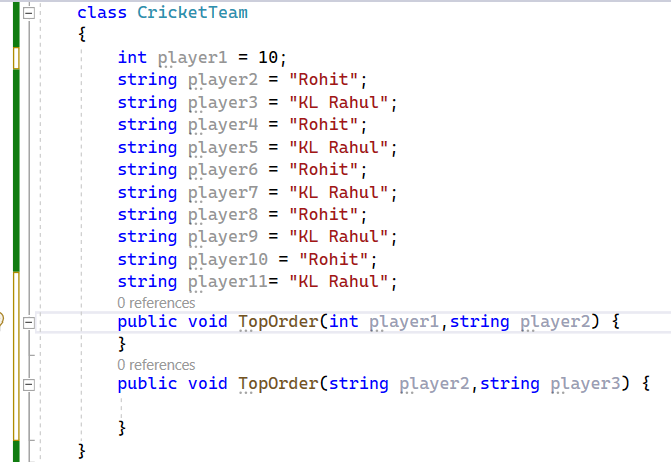
Inside a class when we have multiple methods with same name and same parameters it is known as method duplication.

Method Overloading.

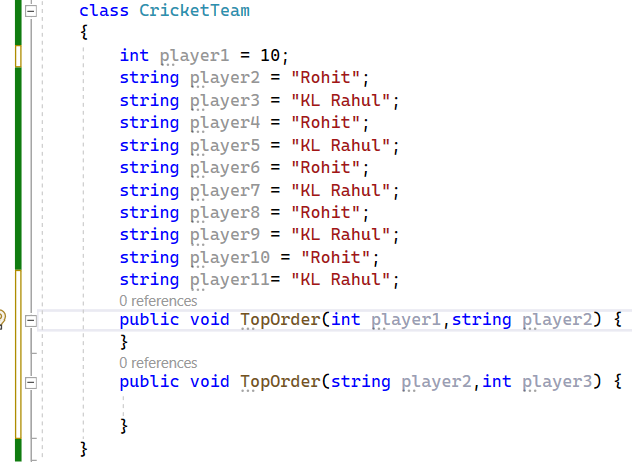
When we have multiple methods with in class and derived class with same name and different in arguments it is known as method overloading.

Rules of method overloading.

1.Number of arguments should be different.

2.If parameters are same then its datatype should be different.

3.If datatypes are also same then sequence of those datatypes should be different.



Method Overriding

Method overriding means a method which present in a super class calling that method from super class to its derived class and changing its implementation/body it is known as method overriding.

Here in C# method overriding is achieved by taking permission from super class by using a keyword called “virtual” and by using “override” keyword in derived class.

Rules of Method overriding:

Return type of method while overriding should be same as super class method.

Method name and number of parameter should be same.

Access modifier should be same.

Private method can’t be overridden.

Static method cannot be overridden.



Polymorphism

Poly means many and morph means forms so polymorphism means many forms.

So, in programing language polymorphism means an object or a method showing different behaviour in different situation of its life cycle it is known as polymorphism.

In Polymorphism we have two types they are:

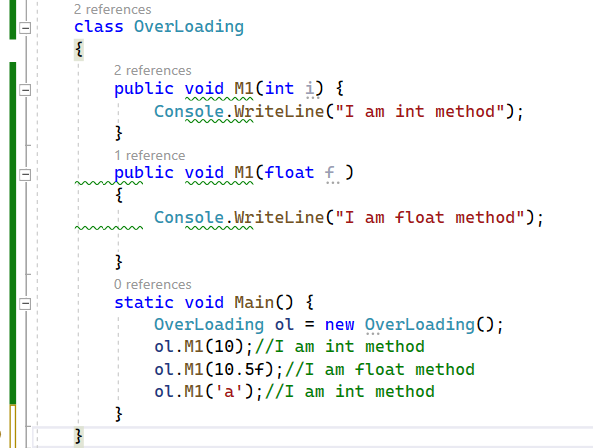
* Compile-time polymorphism.
* Run-time polymorphism.

Compile-time polymorphism:

it is nothing but method implementation is getting binding to its method declaration at the time of compilation it is known as compile time polymorphism.

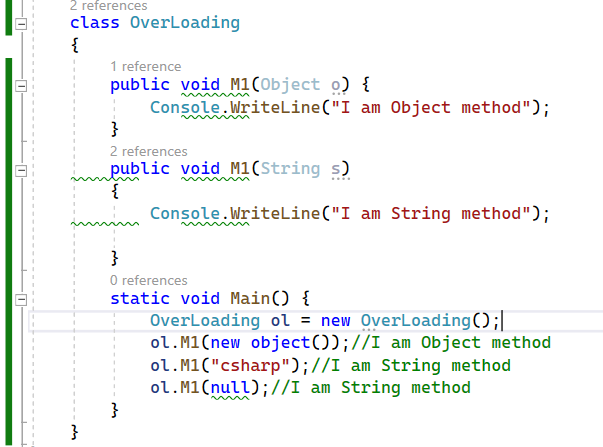
Compile time polymorphism can be achieved by using method overloading.

Case 1



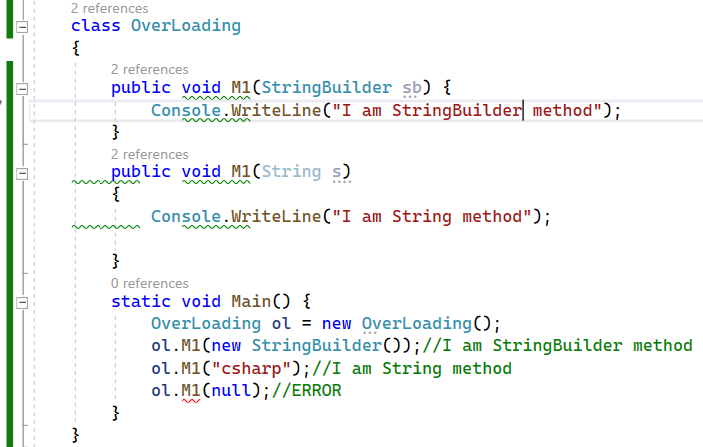
In the above image the M1 method implementation which is taking 10 or 10.5 or char ‘a’ is getting bind to its method declaration at the time of compile time it is known as compile time polymorphism.

Case 2



In the above image the M1 method implementation which is taking object or string and null is getting bind to its method declaration at the time of compile time it is known as compile time polymorphism.

Case 3



In the above image the M1 method implementation which is taking “StringBuilder” or “String” and “null” here null value is not getting bind to any method it is giving error at compile time it is known as compile time polymorphism.

Run-time polymorphism:

it is nothing but method implementation is getting binding to its method declaration at the time of runtime it is known as run time polymorphism.

Run-time polymorphism can be achieved by

a. Inheritance. b. method overriding. c.Up-casting.



Here in above image method implementation is getting binding to method declaration at the time of compilation but while runt-time the output is given from its derived class. It is known as run-time polymorphism.

Note : We cannot declare any members private, protected, protected internal, private protected under namespace that means class, struct, enums and interface cannot be declared with given modifiers.

Abstraction

Abstraction is nothing but hiding the internal implementation and showing what is necessary for users.

Abstract class: Abstract class is nothing but similar to concreate class with an added advantage of

Having abstract methods.

What is abstract method?

Abstract method is nothing but a method declared without implementation, this abstract method implementation is given in derived class that means overriding the method in derived class.

Syntax

Abstract <ReturnType> MethodName();

Points on Abstract class.

No need of declaring “virtual” keyword for overriding the abstract method, but “override” keyword is mandatory in sub class.

We can have private data (variables) and private function(methods) members in abstract class.

We cannot declare private abstract members inside abstract class.

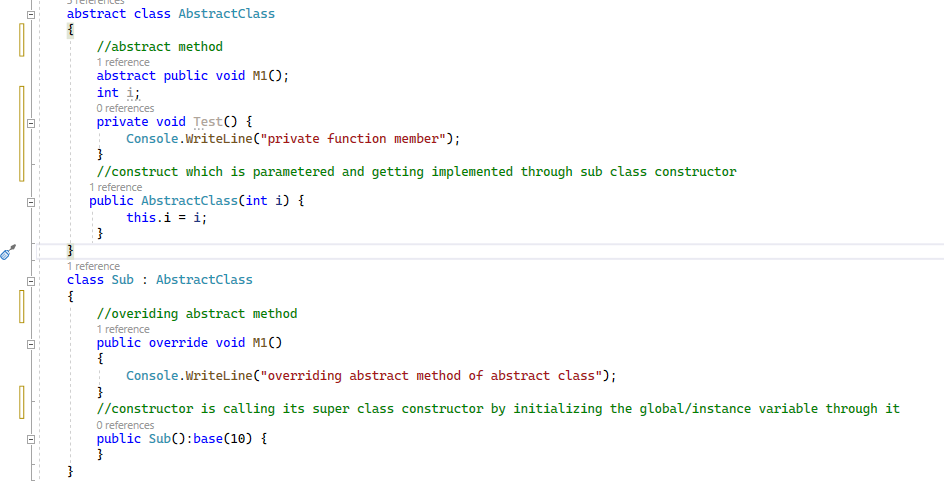
We cannot create object of abstract class.

We can have constructor inside abstract class.

Then what is the use of constructor inside abstract class when we cannot create object of the class?

Use of constructor inside abstract class is to initialize global variables/ Instance variables using inheritance, so for inheritance the constructor is important (constructor chaining) so we have constructor inside abstract class.

If we don’t have constructor, then we cannot implement abstract method.

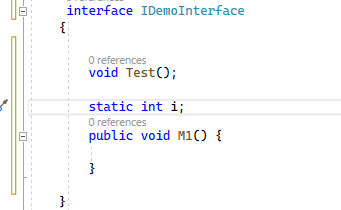


Interface

Interface means it is similar to class but it can contain only abstract method, concrete method and static variable but not constructor.

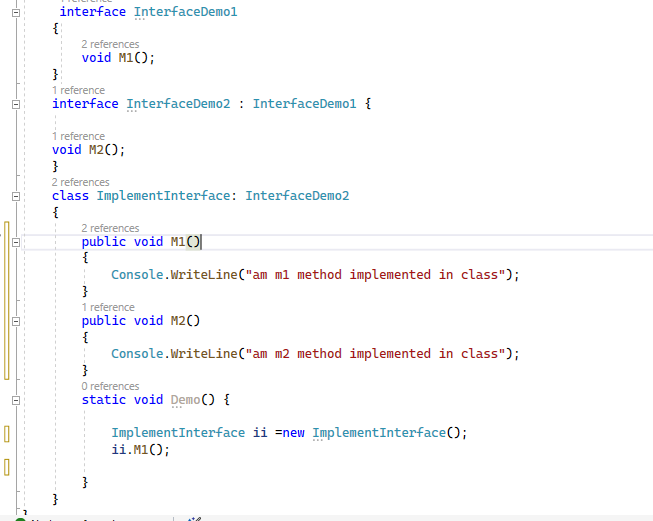
Constructor is not present in interface it is a major reason that we can achieve multiple inheritance using interface.

Here in C# while declaring interface always the interface name should start with Capital “I” indicating it’s an interface.

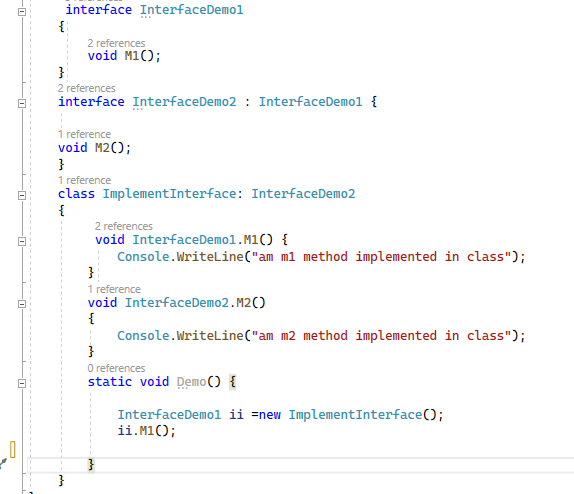


Using and accessing interface in two ways.

Implementing and accessing abstract method of interface default way.



Implementing and accessing abstract method of interface explicitly



Achieve 100% abstraction and multiple inheritance using interface.

