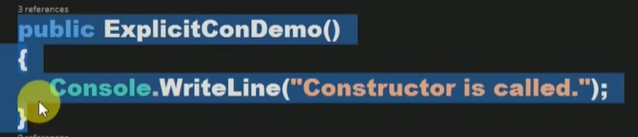
**C#**

**Constructor**

* Every class requires constructor to be present init if we want to create the instance of that class
* Every class contains an implicit constructor if we not define it explicitly and with the help of that implicit constructor instance of class can be created.
* Implicit constructors are Public constructor.
* What is the need of defining explicit constructor again when we already have implicit constructor? Answer: implicit constructor of a class will initialize variable of a class with same value even if we create it multiple time.
* Types of Constructor:

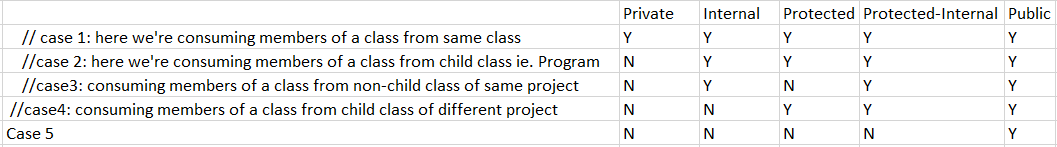
1. Default or Parameter less constructor- if a constructor method doesn’t take any parameter. example below the highlighted has no parameter hence it psrsmrter less or default constructor. Implicit constructors are always Parameter less constructor 
2. Parameterized constructor: if constructor defined with parameter then we can call it parameterized constructor and these parameters can be defined by programmer only to which we call explicit constructor
3. Copy constructor: if we want to create multiple instances with the same value then we use this copy constructor, in a copy constructor the constructor takes the
4. Static constructor

**Access specifires:**

It’s a special kind of modifiers using which we can determine the scope of a type and its members.

***Important note****: You cannot declare class as private, protected, protected-internal. The default scope of a class is internal*

1. Private- the method is only accessible only within class it was defined (the default scope of every member is private. You cannot declare class as private)
2. Internal- if you declare member or a class as internal, its accessible only within the project from child class and non child class also. It cannot be accessed out of the project.
3. Protected-members declared as protected in a class are only accessible only within the child class, not accessible from non-child class.
4. Protected internal-
5. Public-no restriction



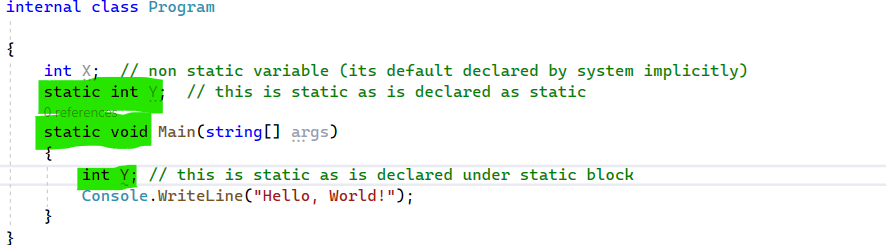
**Variables in classes**

Variable is used to store data. There are 4 types of variable

1. Non Static – when variable is declared implicitly(not declared by programmer)

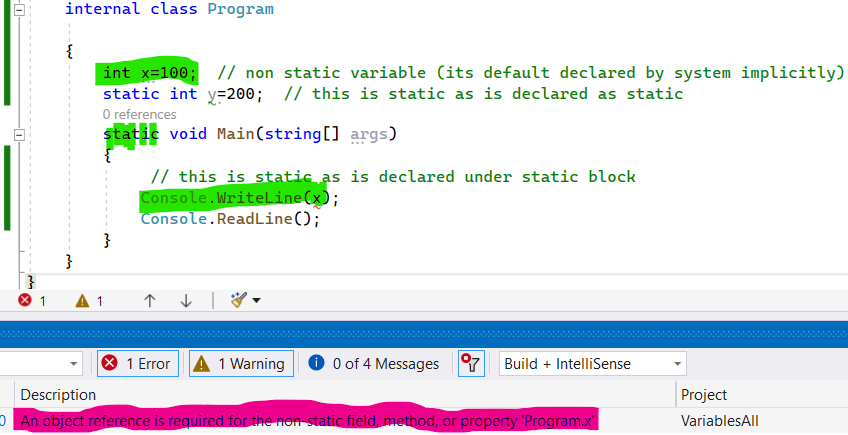


1. Static- if variable is explicitly declared using a static modifier. If a variable is declared in static block then that variable is static. Whereas rest of the other is non static.



***Important: to print the value of static variable we don’t need to create the instance in the main block as the static variable initialized immediately once the execution of a class starts.***

***But to print the value of non-static you have to create an object, which means we need to create the instance to access the value of the variable. The memory of non-static variable gets allocated only if we create instance. See below***



***In the life cycle of a class a static variable is initialized one and only one time, whereas instance variables are initialized 0 time if no instance is created and N number of times if instance is created***

**Constance**: Declared by using keyword **CONST** .These variables can’t be modified once declaration is done .It ***must*** to initialize at the time of declaration only. Instance is not required to be created for Constant and static variables .The only difference between const and static is: Static can be modified whereas constant cannot be modified (Reason: Because we assign the value of constant variable at the time of declaration

**Read-only-** if variable is declared by using readonly keyword then we call it as readonly variable and it cannot be modified like constant but after initialization. Its not compulsory to initialize the readonly variable at the time of declaration, they can also be initialized under constructor. The behaviour of readonly variable will be similar to the behaviour of non-static variable, that is initialized only after creating the instance of class and once for each instance of the class created.

**Difference between Variable Instance and Reference**

**Class**: it’s a user defined type (for example if we talk about string, string is data type but how it became data type, how its developed? String is a pre defined class in our library. So every data type is user defined type).

We cannot consume a class directly. For that we have to create a copy.

For example:

Int = 100; // this is a wrong statement as there is no copy created for int. always remember int is only a blue print for your data. It doesn’t have any memory allocation.

So the correct statement is:

Int **i**=100; // here is a copy of data type where memory stored.

A **variable** of a class is a copy of that class which is not initialized. Example: *String F;* here f is not initialized hence its a variable

An **Instance** is a copy of the class that is initialized by using the **new** keyword.

**Reference**

Reference is a copy of a class which is created using existing instance, reference will not have memory it’s a pointer to an instance.

Modification made through instance will reflects to referenced members.

Program p = new program();

Program p2 = p; // here p2 is a pointer to instance P which is using P’s memory.

**Inheritance IN C#**

It’s a mechanism of consuming the members of one class in another class establishing parent/child relationship between the classes.

**Example:**

Class A

{

-Members

}

Class B:A

{

-consuming the members of A from here

}

A is Parent class or Base or Super class

B is child or derived class

In inheritance child class can consume members of parent class **except** the **private members**

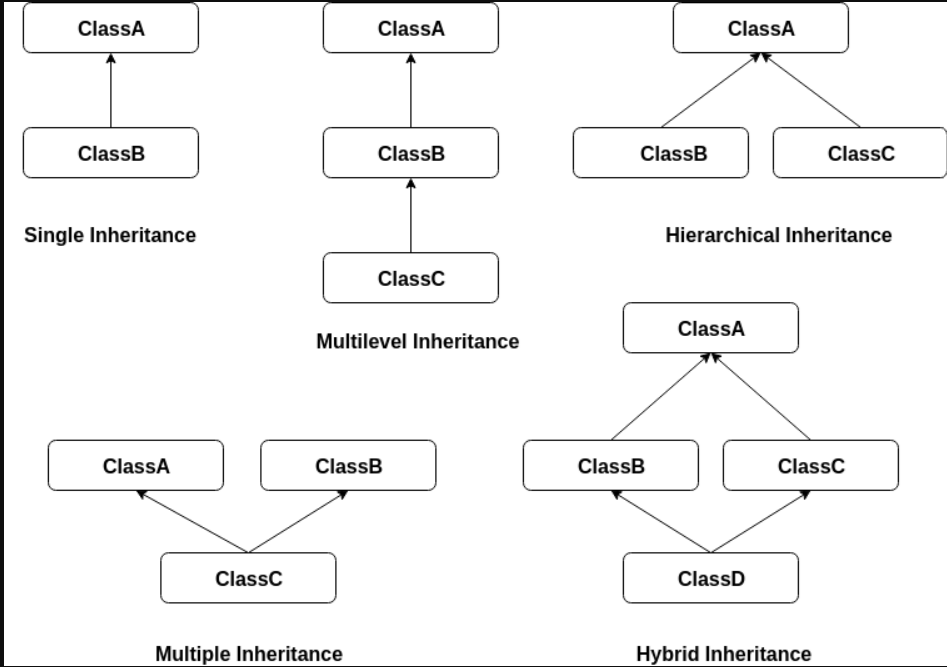
Few Important thing to remember:

1. Parent classes constructor must be accessible to child class, otherwise inheritance will not be possible.
2. Execution always starts with parent class constructor
3. Child class can access members of parent class but parent classes can never access any member of child class
4. We can initialize a parent class variable by using child class instance to make is a reference.
5. Every class that is defined by us or pre defined in the libraries of the language has default class ie. **Object** class of system namespaces. The default parent class is **Object** so the members of object class is **Equals, GetHashCode, GetType, ToString** these are accessible from anywhere.

**Types of inheritance**

Number of parent classes a child can have or number of child classes a parent can have.

1. Single
2. Multi-level
3. Hierarchical
4. Hybrid
5. Multiple



In C# we don’t have support for multiple inheritance through classes, we are only provided single inheritance through classes.

So we are able to use Single, Multilevel and Hierarchical. That is, we’er able to use the inheritance with single parent classes.

1. In the first point we learnt when ever child class instance is created, child class constructor will implicitly called its parent class constructor but only if the constructor is parameter less, where as if the constructor of parent class is parameterized child class constructor cannot call implicitly its parent’s constructor, so to overcome the problem it is the responsibility of the programmer to explicitly call parent classes constructor from the child class constructor and pass values to those parameter. To call parent’s constructor from child class we need to use the **base** key.

**Entity**: its living or non living object associated with a set of attributes.

Suppose **Employee** is an entity then EmpName, EmpAddress, EmpId, EmpSalary is attribute of employee **entity**

Whenever we’re developing any software we’ve to follow certain steps:

Step1: identify the entities that are associated with the application we all developing

Example:

In School application Student, teachingStaff, nonteaching staff,

Step2: identify the attributes of each and every entity.

**Student TStaff NTstaff**

-id -id -id

-Name -name -name

-Address -Address -address

-Phone -phone -Phone

-Class -designation -Designation

-Marks -salary -salary

-Grade -qualification -Dname

-Fees -subject -Mgrid

Step3- identify the common attributes of each entity and put them in hierarchical order. In above table Id, name and address are common attribute. However, to reduce the coding efforts we can use the important characteristic of inheritance ie. Reusability . and we can eliminate duplicity.

For above example after identifying the common attribute I’ll put them in parent class so I can use them anytime by calling that parent class using inheritance.

**Public class Person**

{

Public int id;

Public string Name, Address, Phone

}

**Public class student : Person** // here I have inherited Person class. Now I can only create attribute which left after id, name, address, phone as I have already inherited from Person class.

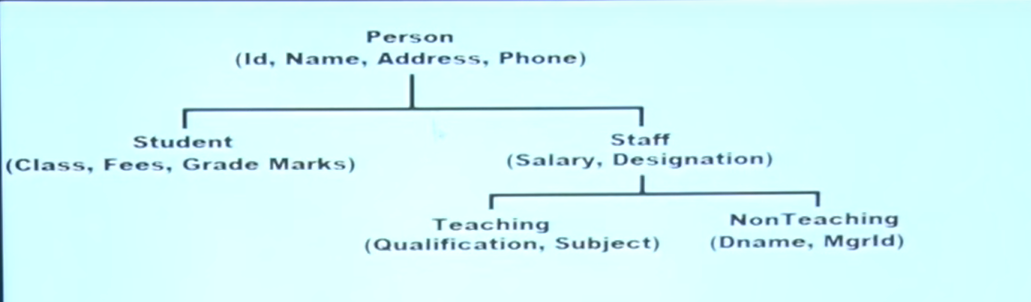
{

Int class;

Char grade

Float marks, fees

}



**Method Overloading**

It an approach of defining multiple methods under the class with same name. however there are few things to remember to avoid ambiguity. We can define same name for multiple methods with the different parameters

Public void Test()

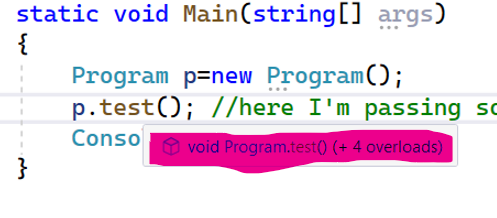
Public void Test(int i) //by parameter

Public void Test(String s) //type of parameter

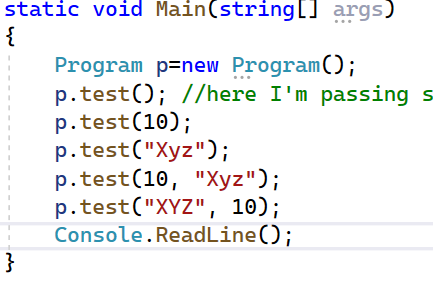
Public void Test(int I, string s) // number of parameter

Public void Test(string s, int i) // order of parameter

After creating instance when call copy of instance you’ll see below highligted message.



In below code, I’m calling methods based on their parameters.



**Why we need Method overloading? Because it comes under one of the principal of OOP ie., Polymorphism : “*behaving in different ways depending upon the input received*”**

**Method overriding**

Its an approach of re-implementing a parent classes method under the child class with exactly the same name and signature.

Eg:

Class1

Test()

Class2:class1

Test()

|  |  |
| --- | --- |
| **Overloading** | **Overriding** |
| We define multiple methods with the same name by changing their parameter | We define multiple methods with the same name and same parameter |
| This can be performed either within a class as well as between parent child classes | This can be performed only between parent child classes can never be performed within the same class |
| While overloading a parent classes method under the child class, child class doesn’t require to take any permission from the parent class. | While overriding a parents method under child class, child class requires a permission from its parent |
| Overloading is all about defining multiple behaviours to a method. | Overriding is all about changing the behaviour of parent’s method of under the child class. |

Note: *if we want to override a parent’s method under the child class, first that method should be declared by using the virtual modifier in parent class*

Class1

Public **virtual** void test() //overridable

Any **virtual method** of the parent class can be overridden by the child class if require by using the override modifier.

Class: class1

Public **override** void test() //overriding

**Method hiding(shadowing)**

Its an approach of re-implementing a parent classes method under the child class with exactly the same name and signature.

In the first case child class re-implements it’s parent classes methods which are declared as virtual, in the second case child class can re-implementing any parent’s method even if the method is not declared as virtual

We can re-implement a parent class method under child class using 2 approaches

1. Method overriding
2. Method hiding/shadowing

We can also call the parent classes methods from child classes by using 2 approaches.

1. By calling the instance of parent class under child class we can call parent’s methods from child class.
2. By using the **Base**  **keyword** also we can call parents method from child class, but keyword like this and base cant be used from static method.
3. A parent class reference even if created by using the child class instance cant’s access any members that are purely defined under the child class but can call overridden members of child class because overridden members which are re-implemented by using the approach of hiding are considered as pure child class members and not accessible to parent’s reference.

**Operator overloading**

This is an approach defining multiple behaviours to an operator and those behaviours will vary based on the operands types between which the operator is used. Example, + is addition when we use numeric operands but when we use string it is called concatenation.

Number + Number = Addition

String +string = concatenation

*A+B: here “+” is operator and “A” “B” both are operands.*

*How computer knows the operations for + is used for addition and – is used for subtraction? Answer is, through libraries.*

Syntax: [<modifiers>] static <return type> operator<opt> (<operand> types>) {logic}

Example: public static int operator + (int a, int b).

*Addition of matrix (logic)*

M1= 20 18 M2 10 8

16 14 6 4

Matrix m1= new Matrix(20, 18, 16, 14)

Matrix m2= new Matrix(10, 8, 6, 4)

Matrix m3= new Matrix (m1.a+ m2.a, m1.b+m2.b, m

1.c+m2.c, m1.d+m2.b)

In above example we’re trying to add 2 by 2 matrix, but what if tomorrow if we’re trying to add more than that, then it would be very complex code.

**Abstract class**

**Abstract Method**: A method without a method body is known as an abstract method, what the method is contains is only declaration of the method.

A class under which we define abstract method is known as abstract class.

To define a method or class as abstract we require to use the abstract keyword on them.

Public void add(int X, int Y); // this is Non-Abstract method there is method body

{

*Method body*

}

Public **abstract** add(int X, int Y); // this is Abstract method as there is no method body and we have to explicitly use the abstract modifier.

**Rule**: *if a parent class contains any abstract method in it, those abstract method must be implemented under the child class.*

**Abstract Class:** a class containing abstract member in it, must and should that class declared ABSTRACT, see below example

Abstract class Math

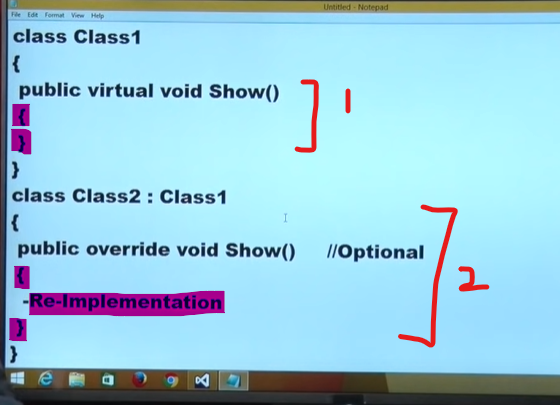
{

Public **abstract** void add(int X, int Y);

}

If a method is declared as abstract under any class then the child class of that class is responsible for implementing the method without fail.

The consent of abstract method will be near similar to method overriding.



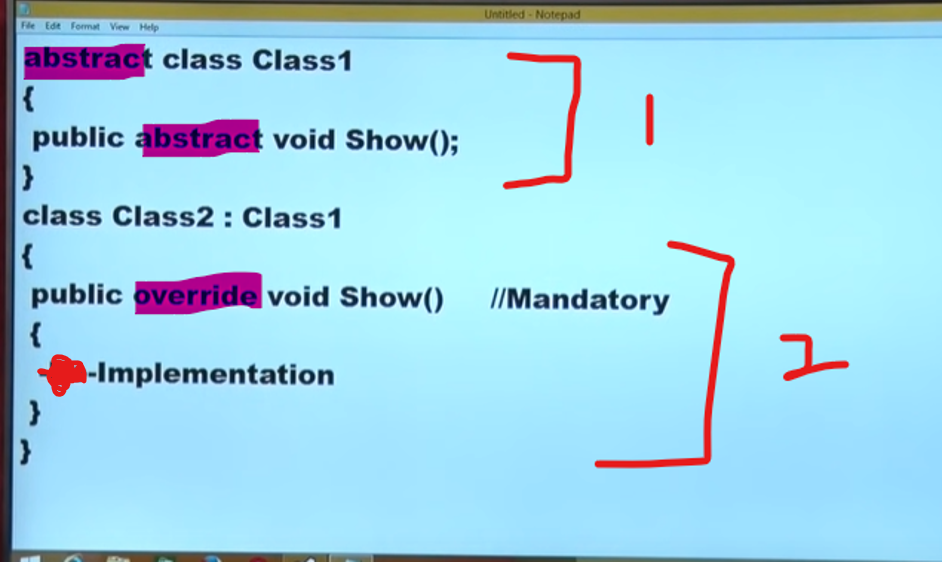
In above screenshot its an example of overriding:

In section 1, the method is implemented as you can see highlighted curly braces

In section 2, the method is re-implemented under highlighted curly braces

In method overriding its **optional** to implement the child method, that means it can be re-implemented.

Below is the example for Abstract class Bookmark Abstract



Section 1, the modifier is abstract and there is no method body under show();

Section 2 , for overriding the parent method we’re following the steps as we did in overriding example above but here implementing the child method is **Mandatory.**

Remember: in abstract example we’re implementing method, not re-implementing.

*“In the case of abstract if at all a parent class contains any abstract method then those method must be implement them in child class by using same Override modifier. This is purely compulsory”.*

**Abstract class can contain Abstract methods as well as non-Abstract methods.**

Child class of parent class:

* Implement each and every abstract method of parent class
* Now only we can consume non-abstract method of parent class

*The parent class contains both abstract and non-abstract method if you want to consume non-abstract method Implement each and every abstract method of parent class*

**Interfaces in C#**

As we all know Class is a user defined data type.

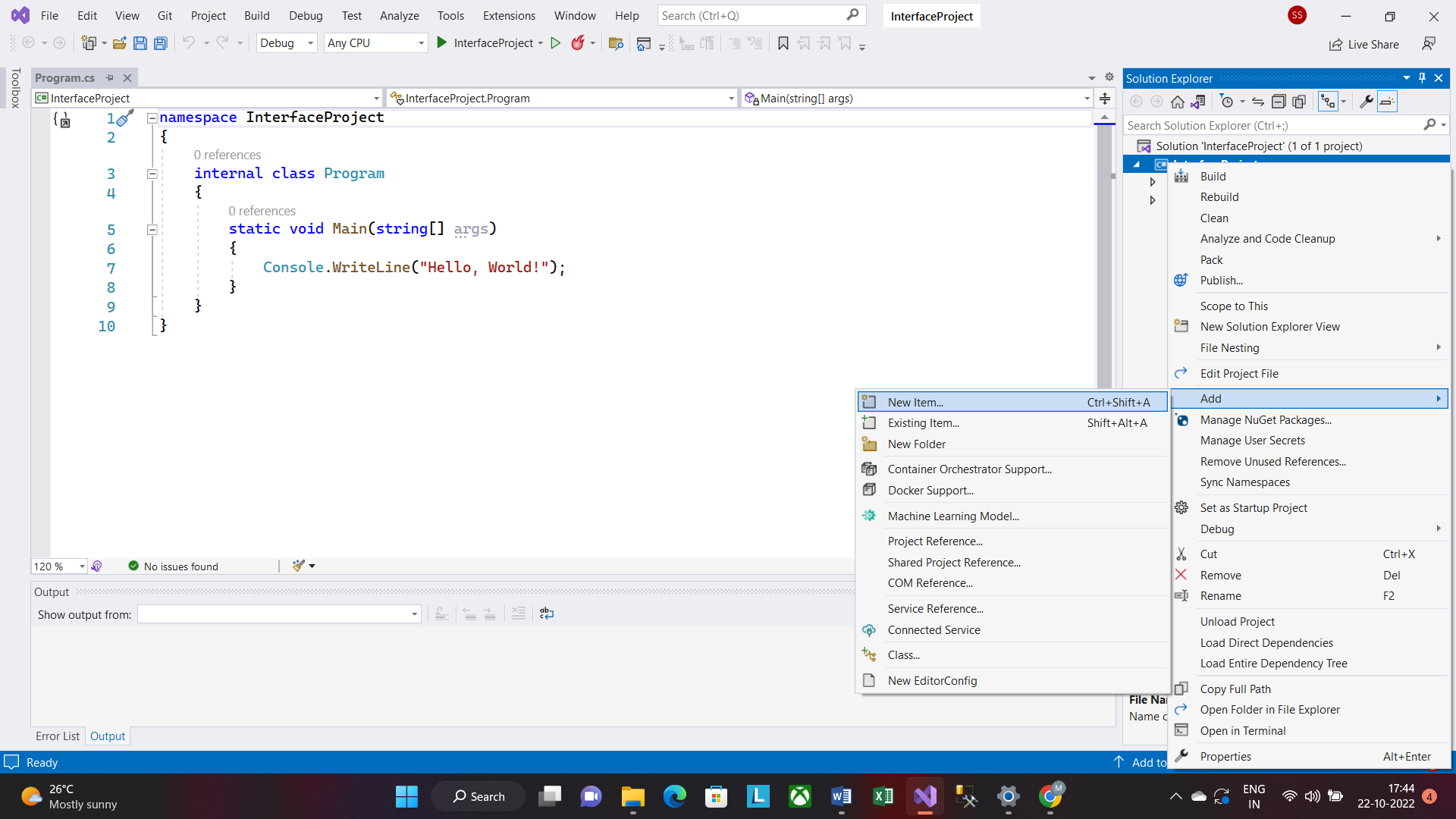
**Interfaces** also a user defined data type, the difference is a class can contain Abstract and non-Abstract methods, however **interface** only contains abstract methods.

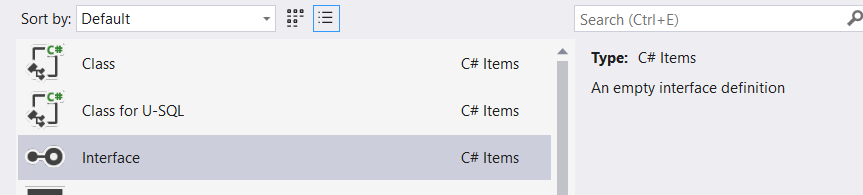
Note: every abstract method of an interface should be implemented by the child class of the interface without fail (Mandatory).

Generally- a class inherits from another class to consume the members of its parents, where as if a class is inheriting from an interface it is to implement the members of its parent.

A class can inherit from a class and interface at a time.

How to add interface in console appl





Syntax:

[<modifiers>] interface <name>

{

-abstract Member declaration here.

}

The **default scope** the members of an **interface** is **public** whereas **its** **private in case of the class**.

By default every member of an interface is abstract so we **don’t require to use abstract modifier** in it again just like we do in case of abstract class. [See this](#AbstractExample)

We cannot declare **variable** under an interface.

If required an interface can inherit from another interface.

Every member of an interface should be implemented under the child class of the interface without fail, but while implementing we don’t require to use override modifier just like we have done in case of abstract class.

**Multiple inheritance with Interface**

Types of inheritance

**Single**, **Multi level**, **Hierarchical**, Multiple, Hybrid.(bolds are supported in C#)

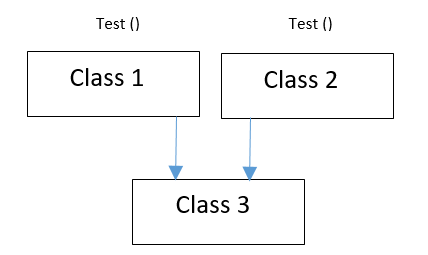
Even if Multiple inheritance is not supported through classes in C#, it is still supported through interfaces.

A class can have one and only once immediate parent class, whereas the same class can have any number of interface as it’s parent. Ie. multiple inheritance is supported through interfaces.

*Why multiple inheritance is not supporter through classes and how its supported through interfaces? Ans: due to Ambiguity its not supported in class*

In below example, there are two methods with same name (ie., Test) , no class can contains two methods with same name(*Ambiguity)*.

Below two classes (1 and 2) are providing the methods to class 3 for **consuming** **the** **methods**



But in Interfaces the interface is containing the methods and requesting class **not to consume** but to **implement the method. Ambiguity will not arise.**

Implementing cannot cause Ambiguity only consuming can.

**Structure in C#**

Structure is also a user defined type. Structure in C language can contain only fields in it whereas Structure in C# can contain most of the members what a class can contains like, fields, methods, constructor, properties, indexes, operator, operator methods.

Syntax:

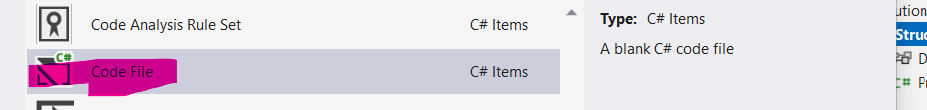
[<modifiers>] struct <name>

{

-define members

}

How to add structure in .net, there are many template like class, interface etc but you wont find structure, hence to add it click on below template



Code file is blank template, for it you can create many template by own

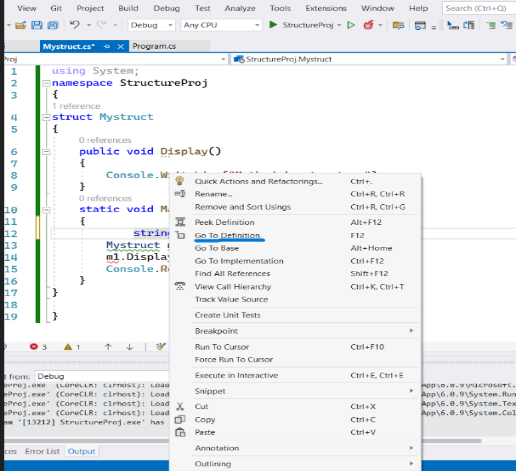
Differences between classes and structure

|  |  |
| --- | --- |
| **Class** | **Structure** |
| Class is reference type (memory allocated managed heap) | Structure is value type(memory allocated stack memory) |
| We use classes for representing an entities with larger volumes of data. | We use structure for representing an entities with smaller volumes of data. |
| New keyword is mandatory to create instance | New keyword is optional |
| Fields of a class can be initialized at the time of declaration | Not possible here. |
|  |  |

*A* *Value Type holds the data within its own memory allocation and a Reference Type contains a pointer to another memory location that holds the real data.*

All the pre defined data types under the libraries of our language which comes under reference type category, example string and object are classes, whereas all predefined data type which come under value type category example: int(int32), float(single), bool(Boolean) are structure.

To check that: type string and right click on go to definition, you’ll show you meta data, see below: string is class





Same way check INT you’ll see meta data for it as struct as below:



Whatever we learnt as value type is structure and reference type is class.

Note:  *if the structure contains any fields then we need to initialize those fields either by explicitly calling the default constructor with the help of new else if we are not using “new” keyword for creating the instance we need to explicitly assign the value to the fields referring it through the instance and assign the value.*

*We can defined any constructor under the class, ie. Either parameterized or parameterless and if no constructor is defined then there will be an implicit constructor, which is default. Whereas in case of a structure parameterless of default constructor is always implicit and cannot be defined explicitly again what we can define a parameterized constructor.*

*If zero const are defined in a class after compilation there will be 1 constructor (implicit) and if we define “n” constructor in class after complication there will be “n” constructor only. Where as in case of structure if we defined “0” constructors then after compilation there will 1 constructor(implicit) if we define “n” constructor after compilation there will be n+1 constructor.*

*Class can be inherited by other classes, whereas structure cant be inherited by other structure ie. Structure doesn’t support inheritance.*

*A class can implement an interface, same as that a structure also can implement an interface.*

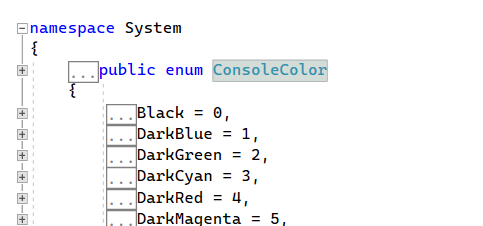
**Enumeration or Enum types in C#**

Enum is user defined type, so it is always better to define an Enum directly under the namespace, but it is also possible to defined a enum under a class or structure also.

Enum comes under Value type category. What is value type category? Click [here](#types)

[<modifier>] enum <name> [:<type>]

To check metadata, right click on consoleColor and select go to definition, then you can see below



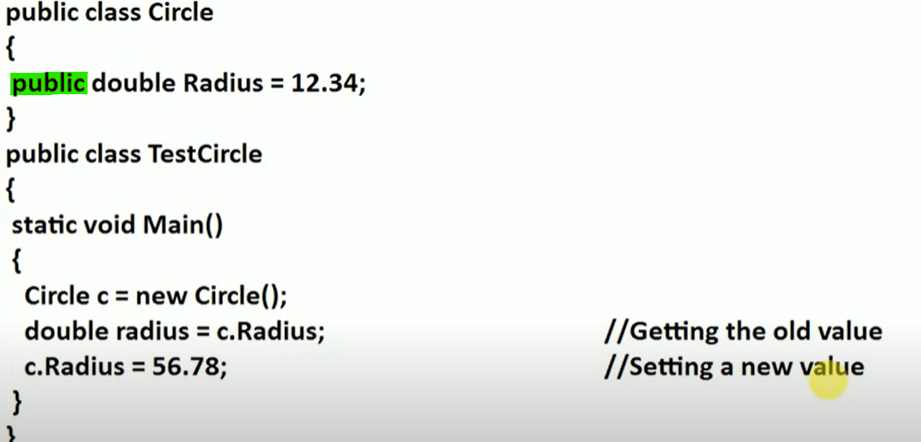
Accepted data types are: byte, short, int, long, uint,ushort, ulong and sbyte

**The data type is byte**

The Advantage of Enum is you’re trying to define a method a property with only set of value and select the value from that list only, we use enum.

**Properties in C#**

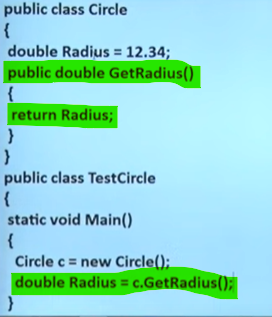
Properties is a class using which we can expose values associated with a class to the outside environment.



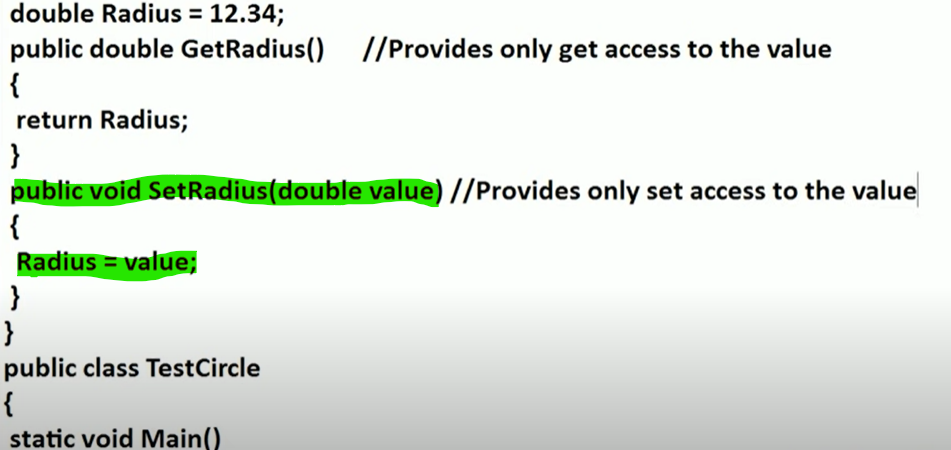
In above picture, we have declared variable with modifier as Public. Why? Because we cannot access because its private [see this](#private) . The **default scope** the members of a class is private. Hence I declared it as public so I can access it anywhere, and also anyone can get and set the value, but here is the problem, once the variable is declared as public anyone can get the value and set the value, we cannot restrict it and have control over this.

Remember one thing: “do not declare the variable as Public” to have the control and not to give anyone the permission to change the value of it.

So how we can access the value after declaring the variable as public?? Answer is by Methods. See below:

here I have created the GetRadius() method and accessing it main method.

In the same way we can set the value , see below:



*A* ***property*** *is basically a* ***combination*** *of two methods* ***GET and SET***

[<modifers>] <type><name>

{

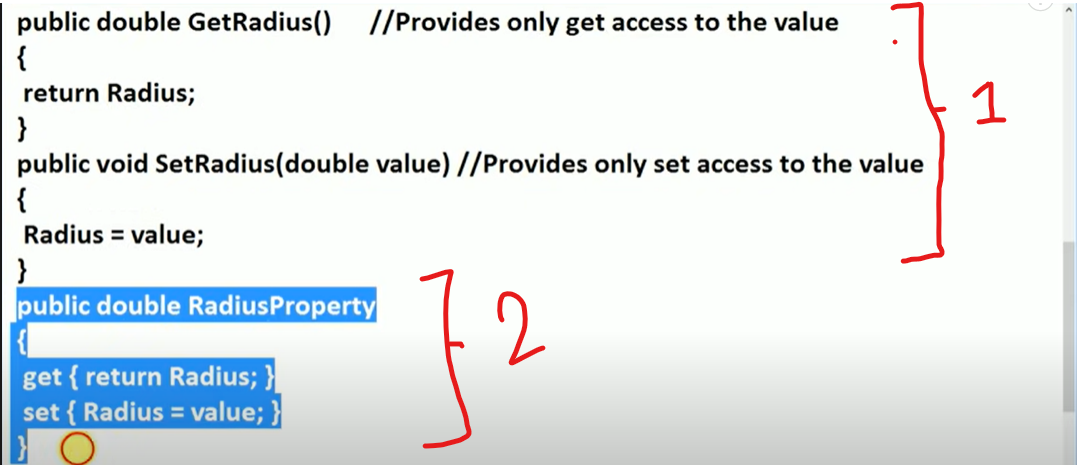
[get{<stmts>}] //get accessor

[set{<stmts>}] //get accessor

}

The advantage is unlike [above](#methods) ,

1. no need to give two different names(getRadius and SetRadius) will use the same name for GET and SET
2. We don’t need to create two methods for get and set. We can perform the action in same block as shown above.
3. No need to declare parameter for method.



The 1st block is [old](#oldway) way of accessing the values outside of a class and 2nd block is [new](#newway) on to which we call **properties.**

How to implement in main method? See below

Static Void main()

{

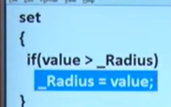
Circle c= new Circle();

Double Radius = c.RadiusProperty;

c.RadiusProprty=56.78;

.}

We can also provide the conditions to properties, see below



**Indexers**

Is member of a class if defined this in class your class starts behaving like a virtual Array.

As we know the **default scope** the members of a class is private. So, how we can consume the members of the class outside. We’ve three option

1. Declare all the member to public- but as we know we cannot do the same as we’ll lose the access.
2. Properties : see [this](#properties).
3. Indexers: it is something like properties but provides the access to the value using index position.

You cannot apply indexing on a class, directly.

For that we need to define an Indexer to the class.

Syntax: [<modifiers>]<type>this[Parameter List]

{

[Get{<stmts}]

[Set{<stmts}]

}

When you define indexer in a class the class start acts like a virtual Array.

**Delegates**

It’s a type safe function pointer

A delegate holds the reference of a method and then call the method for execution.

Delegates allow methods to be passed as parameters

Till now we used to call methods in 2 ways,

For Non-static method: create instance in main block.

For static method: call it by class name.

The other way calling method by delegates:

To call a method by using delegates we have 3 steps: *refer C# program*

**Step 1. Define a delegate**. <modifier> delegate void|type <name>(<parameter list>)

public delegate void AddDelegate(int x, int y); // *remember: return type and paramerts should exactly match with the parameter of method. (compare delegate method with below method)*

Public void AddNums(int a, int b)

Public delegate string Saydelegate(string str)

Public static string SayHello(string name)

Delegate is reference type hence we have to create instance. We define delegate in namespaces

**Step 2. Instantiating the delegate:**

**Step 3: calling a delegate**

**Multicast Delegates**

A delegate will be holding reference of more than one method and it can be called with the help of same delegate.

**Anonymous Delegates**

Amethod without a method body which can bound delegate and can be called

Without binding a named method to the delegate you can bind a code block to the delegate.

What is advantage: less code writing.

Don’t use Anonymous it is suggested to do when only less coding is there, in huge coding programme no need to use this consept.

**Generic Delegates**

Predefined delegates.

Func, Action, Predicate

**Lambda expression Delegates**

Lambda expression is short hand for Anonymous Delegate, it uses to simplfy the anonymous method.

**Exception Handling**

**Compile time error**- error by human mistake for example, program will not run I semicolon is missing or curly braces is missing etc, this occur at the time of compilation

**Missing resources exception:**  this happens if required file or DB is missing.

**Runtime Error:** these are error which are occur not at the time of compilation but at the time of execution.

Reasons for Runtime error:

-wrong implementation of logic

For Example if we need to create array and we have write the code as below:

**Int[] arr={10,20,30,40,50}**

**For (int i=0, i<=arr.length, i++) //**here we are counting from 0 and array length is 5

**{**

**Console.writeline(i)**

**}**

**Console.writline(“end of programme”)**

In above code the implementation is wrong as we have provided 5 values in array list and we’re counting from 0 to 5, since total 6 values will be retrieved hence the implementation is wrong. As there are only 5 values and we’re asking to retrieve 6.

-wrong Input supplied

Writing a program asking to enter integer value, bur user enters wrong value like double or string, so that is wrong Input.

-Missing required resource

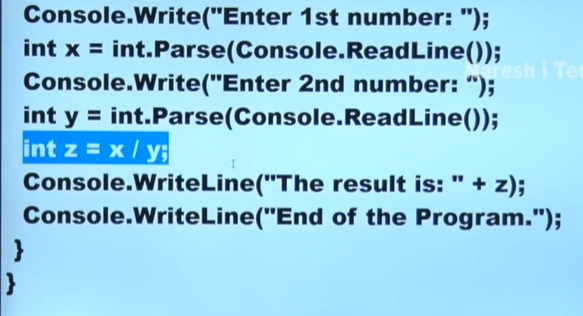
If required file is deleted, or if required database is not there or required servers are shut down.

***Runtime errors are dangerous, they going to cause abnormal termination, wherever the error occurs rest of the code will not execute***

Who is responsible for runtime errors? Ans- Exceptions

Exceptions are actually a class, there are so many classes available in our base class libraries and these classes are responsible for abnormal termination of the program.

In below code, we’re going to divide two numbers x and y. If I gave two values 10 and 0 then there will be exception called divide by 0 exception.





And it will not show further lines ie.



Because it is terminating at int Z=X/Y; above two lines did not execute.

In above program, **CLR** monitored and checked what the problem is then it concludes that we’re trying to perform invalid operation.

So here the dividedbyzero class associated with this exception and abnormal termination happens.

Whenever there is runtime error CLR will create instance of a matching exception class and throws it and instance which is thrown by CLR will immediately perform an abnormal termination.

***Whenever the runtime error occur, to control these abnormal termination we have a mechanism called exception handling.***

**Advantage of exception handling:**

1. Abnormal termination stops so that statements that are not related with the errors can be executed.
2. We can display user friendly errors message to the end users so that we can describe about the error
3. We can perform corrective action that can be useful for resolving the issue.

To handle these action we need to add our code under a block as below.

Try

{

Stmts’s which will cause runtime error

Stmts’s which does not require execution when runtime error occurred

}

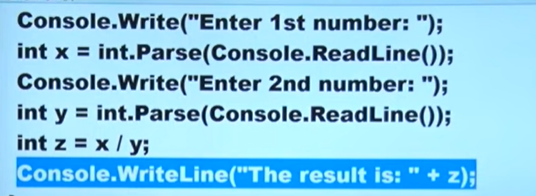
Catch(<exception class name)>) <variable>

{

Stmts which should execute only when there is runtime error

}

See below code: in below scenario highlighted statement causing a runtime exception and does not require execution when runtime error occurred hence we have to put it in **try** block and the



**Finally**

Finally block execute when exception occur and exception not occur. Finally block execute compulsory, it will run at any cost.

User: if you are developing application and you want a message/code should execute at any code then put that under finally block.

We can also create our own exception.

For example: if you want to create an exception for divide by odd number is not allowed, here CLR will not catch this exception as divide by odd not universal rule as divide by zero.

Hence for divide by odd rule should be created by us.

So creating this exception we can create the instance for SystemException

As we already know Exception is parent class wherein we have two child class

-SystemException (all predefined exceptions instance)

-ApplicationException

As we know CLR will create instance for relevant class and throws it in case of any exception. So in the same way we will create instance for user defined exception

To perform the same we are creating instance:

ApplicationException ex = new ApplicationException (“<Error Message>”);

Throw ex;

Or

Throw new ApplicationException(“<Error Message>”);

**Multithreading in C#**

A thread is a unit which executes a code under an application, so every application has a logic to execute that logic thread comes into the picture. Every application has by default one thread that executes logic and that is called as main thread. **Every application has by default a single threaded model**

So, the drawback is, in the single threaded model the action will completed one after the other.

In this scenario suppose if any programme take more than expected time.

For example: if my method is connected with DB and trying tom load and perform some action and come back, but unfortunately the DB is busy responding, so my method is waiting there to respond and taken some time and after some time the DB got responded so communication happened and came back. But just suppose the DB taken 10-20 seconds to responds and because of that method 2 is getting delayed there, so we are losing time because its sequential execution.

To overcome of this issue we’re provided with Multi-Threading. So under a process we can have multiple threads where each thread is trying to perform a task, based on the time sharing all the thread going the run equally. For example, if we try to run 3 methods and 1st thread is taking time to execute so the time will not get wasted OS will execute the other 2 threads, so time will not be wasted.

This is the advantage of **Multi-Threading.**

-It will run on time sharing manner

-maximum utilization of CPU resources

**Use of** **join**: the calling thread will not exit program until the all the other threads in the program are exiting from the program. Join is overloaded method

**Locking thread:** if we are dealing with multiple thread and we want one thread accessing the code and after that code finishes the work then only the control goes to the second thread and so on…

**Thread Priority:** to set priority between threads.

Lowest

Below Normal

Normal

Above Normal

Highest

**Collections in C#**

There are three things never possible with array.

1. Increasing the size
2. Inserting the values into the middle
3. Deleting or removing the value from the middle.

To overcome with this problem we just depend on the **collection**. A **collection** is nothing but a dynamic array. A collection is capable of storing multiple value but it also has above three mechanism in it, so collection are auto resizing, inserting values into middle, deleting/removing value. Since we can add object in array list, hence added value can be anything like, int, string, bool, etc

In .Net Stack, Queue, linked List, Sorted List, Array List, Hast table are already available as classes under System.Collections.

Difference between array and arrayList

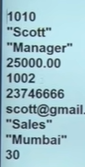
|  |  |
| --- | --- |
| Array | ArrayList |
| Fixed Length | Variable Length(size is not fixed, auto resizing) |
| Not possible to insert item | We can insert item from middle |
| Not possible to delete items | We can delete items from the middle |
| Type safe(can provide only specific types of variable) but fixed length | Auto resizing but not type safe (can provide any types of variable) |

Array list has a feature called Add() this will add item in array list at the end of the Arraylist.

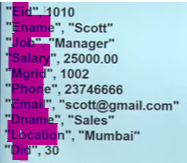
A capacity is a property which tell the number of items that can be stored under a collection.

**Hash Table collection:**

It’s very hard to remember the index value in large number of data, to overcome this problem we have Hash table. For example, if I want value of email address I have to remember the key ie. 8



A hash table is also like array list which is capable of auto resizing itself but it is going to store the value in key value combination here the advantage is the key value is user defined. For example : in below scenario I gave key which is user defined.



In hash table the values are stored in hash code and to each key the hash code is allocated. Hence we won’t get the output in given sequence manner.

**Generic Collection (capable of storing specific value):**

As we know array is type safe but fixed length and array list is auto resizing but not type safe.

If we think to add both good features of array and array list it should be type safe and auto resizing, then C# has provided a facility called **Generic collection**.

For Example:

Public class Customer // this is new type what we have been defining

{

Public int Custid{get; set; }

Public string CustName{get; set; }

Public double CustBalance{get; set; }

}

List<int> li = new List <int>(); // this is capable of storing integer type value

List<string> ls = new List<string>(); this is capable of storing string type value

List<Customer> Customer = new List<customer>(); // this is capable of storing customer type value

*System.collection.generic* is name space that we are going to use for generic collection

**Dictionary**

In arraylist we had Hash table, here in collection we have Dictionary.

**IComparable** Interface in C# to **sort elements**. It is also used to compare the current instance with another object of same type

The Sort () is overloaded method hence we can perform few more types of sorting as below

Sort() // no parameter: simple type

Sort(<instance>)// it takes a Icomparer as a parameter

**IEnurable**

IEnumerable helps to iterate over a collection without knowing the actual type.

**LINQ to SQL**

LINQ- language integrated query

It’s a query language that is introduced in .net 3.5 framework for working with relational DB ie. SQL Server.

SQL is query language using which we can access data from relational DB, Exactly Microsoft has in its .Net provided LINQ, its also a query language with the help of that we write queries on a wide variety of data source.

LINQ to SQL is not only querying the data but also allows us to insert update delete the data.

We call them as CRUD

CRUD- create (Insert), Read (select), update, delete. (We can also call them stored procedure)

Already there is a language known as SQL using which we can interact with SQL server with the help of ADO.Net

Using SQL we’re interacting with SQL server

* Runtime syntax checking of SQL stmts(here the verification of code will be done by DB engine) it will put extra burden on DB in case of large amount of data and impacts on performance.
* Not a type safe
* No intelligence support
* Debugging of SQL stmts is not possible
* Code is combination of object oriented and Relational

Using LINQ we’re interacting with SQL Server

* Compile time syntax checking, here the verification part will be done by C# compiler hence it will not put burden on sql so performance will not be impacted.
* Type safe
* Intelligence support available
* Debugging of Linq SQL is possible
* Pure Object oriented code.

LINQ to object: using this we can write queries on arrays, collection etc.

LINQ to DB: using this we can write queries on DataTables, relational DB tables

* Linq to ADO.net
* Linq to SQL
* Linq to Entities

LINQ to XML: using this we can write queries on XML files

To work with LINQ to SQL first we need to convert all the relational objects of DB into object oriented types and this process is known as ORM(object relational mapping)

Find below conversion

|  |  |
| --- | --- |
| **Relational Objects** | **Object Oriented types** |
| Table | Class |
| Columns | Property |
| Rows or Records | Instance |
| Stored procedures | Methods |

To perform ORM we’re provided with a tool called **OR designer**