* What is C#?

C# is a modern, general-purpose, object-oriented programming language developed by Microsoft and approved by ECMA & ISO.

European computer manufactureres associataion.

International Stanadered organization

**Example:**

using System;

namespace HelloWorldApp

{

class HelloWorld

{

static void Main(string [] args)

{

Console.WriteLine("Hello World");

}

}

}

#using System

-> The using keyword to include the System namespace in the program. A program generally has multiple using statement.

#namespace HelloWorldApp

-> A namespace is a collection of classes.

#class HelloWorld

-> It just a class decleration. Classes senerallw cantainse miltinle methndse. Method defines the behaviour of the class.

'#static void Main(string [] args)

-> This is the entry point for all c# programs. The Main method states what the class does when executed.

* static-> It is a keyword which means object is not required to access static member. So it saves memory.
* Void -> It is the return type of the method. It does't return any value. In such case, return statement is not required.
* Main-> It is the method name. It is the entry point for all C# programs. Whenever we run the C# program, Main() method is invoked first before any other method. It represents startup of the program.
* string [] args-> It is used for command line arguments in C#. While running the C# program, we can pass value.
* Console.WriteLine("Hello World") 🡪 Console is the class defined in system namespace. The WriteLine() is the static method of Console class which is used to write the text on the console.

#Important Points:

1) C# is Case sensitive

2) All statement and expression must end with a semicolon(;)

3) The program execution starts at the Main Method.

4) Unlike JAVA, program file name could be different from the class name.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* **#Class**

-> A class is like a blueprint of a specific object.

**Example:**

Ferrari is an object of the Luxury car type.

-> Generally a class decleration contains only keyword class, followed by an identifier.

**Example**: class HelloWorld.

**Components of Class**

(Depends on Requirements):

1) Modifiers (DR)

2) Keyword class

3) Identifier

4) Base Class or Super class(DR)

5) Interface(DR)

6) Body{}

**#0bject**

-> It is the basic unit of OOP and represents the real life entities. An object consists of 3 things.

1) State

2) Behaviour

3) Identity

**1) State/Attributes**

-> It is represented by the attributes of an object. It also represents the properties of an object.

**2) Behaviour**

-> It is represented by the methods of an object. It also reflects the response of an object.

**3) Identity**

-> It gives a unique name to an object and enables one object to interact with another object.

Example: Carly, Weekly.

**#Constructor**

-> A Constructor is a special method which gets automatically called/Invoked whenever an object/Instance is created.

**-> Important Points:**

1) Constructor of a class must have the same name as the class name which it resids.

2) A constructor can not be abstact, final.

3) A Constructor doesn't have any return type not even void.

4) A class can have any number of Constructor.

5) Access modifiers can be used in Constructor decleration to control it's access.

6) Within a class, you can create only one static Constructor.

7) A,static Constructor can not be a prameterized Constructor.

**-> Types of Constructor**

1) Default Constructor

2) Parameterized Constructor

3) Copy Constructor

4) Private Constructor

5) Static Constructor

**1) Default Constructor:**

-> A Constructor with no Parameter.

**2) Parameterized Constructor**

-> A Constructor having at least one parameter.

**3) Copy Constructor**

-> This Constructor creates an object by copying variables from another object. It's main use is to initialize a new instance to the values of an existing Instance.

Book b1 = new Book(10,20);//a=10,b=20

Book b2 = new Book();//a=0,b=0;

b2=b1

//a=10,b=20

**4) Private Constructor**

-> If a constructor is created with private specifier

-> It is not possible for other classes to derive from the class.

-> It's not possible to create an instance of the class.

**5) Static Constructor**

-> If a constructor is created with static keyword.

-> It is initialized static fields or data of the class and to be executed only once.

-> It can't be called directly.

-> When it is executing then the user has no control.

-> It doesn't take any access modifier.

-> Within a class, you can create only one static Constructor.

-> A static Constructor can not be a parameterized Constructor.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**#Destructor**

-> Destructors in C# are special methods inside the class used to destroy instances of the class when they are no longer needed.

**-Important Points:**

-> The Destructor is called implicitly by the .Net Framework Garbage Collector and therefore programmer has no control as when to invoke the destructor.

-> A Destructor is unique to its class. There cannot be more than one destructor in a class.

-> A Destructor has no return type and exactly same name as the

class name.

-> It is distinguished apart from constructor because of the Tilde symbol(~) prefixed to its name.

-> A Destructor does not accept any parameter and modifiers.

-> It can not be overloaded or Inherited.

-> It is called when the program exists.

**#Access Modifier**

-> Access Modifiers are the keywords that define the accessibility of a member, class or datatype in a program.

**-> Types of Access Modifiers(4):**

1) public

2) protected

3) private

4) internal

**-> Types of Accessibility Level(6):**

1) public

2) protected

3) private

4) internal

5S) protected internal

6) private protected

* **Types of Accessibility Level(6):**

**1) public**

-> There is no restriction on accessing public members.

**2) protected**

-> Access is limited to within the class definition and any class that inherits from the class.

**3) private**

-> Access is limited to within the class definition

**4) internal**

-> Access is limited exclusively to classes defined within the current project assembly.

**5) protected internal**

-> Access is limited to the current project assembly and types derived from the containing class. All members in current project and all members in derived class can access the variables.

**6) private protected**

-> Access is limited to the containing class or types derived from the containing class with the current assembly.

\*\*\*\*\*

**Default Access Modifier** :

1) Class -> Internal

2) Class Member -> Private

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**PROPERTIES**

* **What is a property in C#?**

-> A property in C# is a member of a class which is used to set and get the data from a data field of a class.

-> The most important point about property in C#, It is never used to store data, it just acts as an interface to transfer the data.

-> We use the property as they are the public data members of a class

-> It has special methods called accessors.

* **What are the Accessors in C#?**

-> The Accessors are nothing but special methods which are used to set and get the value from the underlying data members.

**-> Accessors are two types:**

1) set Accessor

2) get Accessor

* **set Accessor**

-> The set Accessor is used to set the value into a data field.

-> The set Accessor contains a fixed variable named "value"

-> Whenever we call the property to set the data, whatever data we are supplying that will come and store in the variable value by default.

* **2) get Accessor**

-> The get Accessor is used to get the data from the data field.

-> Using this get accessor you can not set the data.|

* **#What are the different types of properties?**

-> The C# .Net supports four types of properties:

1) Read-only property

-> The Read-only property

2) Write-only property

3) Read-Write property

4) Auto-implemented property

**1) Read-only property**

-> The Read-only property is used to read the data from the data field

-> Using this property you can not set the data into the data field

-> The property will contain only one accessor -> get accessor

**2) Write-only property**

-> The Write-only property is used to Write the data into the data field.

-> Using this property you can not read the data from the data field.

-> The property will contain only one accessor -> set accessor

**3) Read-Write property**

-> The Read-Write property is used for both read the data from the data field and write the data into the data field.

-> The property will contain both accessor-> set and get.

**4) Auto-implemented property**

-> If you do not have any additional logic while setting and getting the data from the data field then you can use of the auto-implemented property.

-> The auto-implemented property reduces the amount of code that we have to write.

* **# What are the advantages of Properties in C#?**

1) Properties will provide the abstraction to the data fields.

2) They also provide the security to the data fields.

3) Properties can also validate the data before storing into the data

fields.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ENCAPSULATION**

-> Encapsulation is the concept of wrapping data into a single unit.

-> It collects data members and member functions into a single unit called class.

-> The purpose of encapsulation is to prevent alternation of data from outside.

* **#How can we implement encapsulation in c#**

1) By declaring the variables as private.

2) By defining one pair of public setter getter methods or properties to access private variables.

* **#Advantages**

1) Data Hiding

2) Increase Flexibility

3) Reusability|

4) Testing code is easy

**#Advantages**

1) Data Hiding

-> The user will have no idea about inner implementation of the class.

2) Increase Flexibility

-> We can make the variables of the class as read-only or write-only depending on our requirements.

3) Reusability

-> It improves the reusability and easy to change new requirements.

4) Testing code is easy

-> Encapsulation code is easy to test for unit testing.

* **Date : 14 JUNE 2021 Lab Task 0.0 Time : 1 Hour 0 min**

1.Write a C# program which asks for a student score.Score is a number from @-100, Translate the score into grade like A+, A, B, C, D&F.

2. Write a C# program that asks the user to type the price without tax of one kilogram of Potatoes, the number of kilograms you want to buy and the tax in percent units. The program must write the total price including taxes.

3, Write a C# program that asks the user to type 2 integers A and B and exchange the value of A and B,

4, Write a C# program that asks the user to type 5 integers and writes the average of the 5 integers. This program can use only 2 variables.

5. Write a C# program that asks the user to type the width and the height of a rectangle and then outputs to the screen the area and the perimeter of that rectangle.