API DOCUMENT

- **Visual Studio 2022 Overview**
  - Visual studio is an Integrated Development Environment developed by Microsoft.
  - It provides support for 36 different programming languages. It is available for Windows as well as for macOs.

- **Different types of windows**
  - The IDE has two basic window types, tool windows and document windows. Tool window include solution explorer, server explorer, output window, error list, the designers, the debugger windows.
  - When you have to view or edit two locations at once in a document, you can split the windows.

- **Solution and Projects**
  - A solution is simply a container visual studio uses to organize one or more related projects.
  - Create a solution
    1. Open visual studio.
    2. On the start window, select Create a new project.
    3. On the Create a new project page, enter blank solution into the search box, select the Blank Solution template, and then select Next.
    4. Name the solution and then select Create.

- **Code Editor Features**
  - The visual studio editor provide many features that make it easier for you to write and manage your code and next.
  - Features are :-
    1. Syntax Coloring
    2. Error and Warning Marks
    3. Brace matching
    4. Selecting code and text
    5. Change tracking

- **Shortcuts**

Task	**Shortcut**
Maximize floating window	Double click on title bar
Maximize/ minimize window	Win + Up arrow / Win + Down Arrow
Close Active document	Ctrl + F4
Start new Project	Ctrl + Shift + n
Open Project	Ctrl + Shift + o
Shown open file list	Ctrl + Alt + Down Arrow
Comment	Ctrl + k, Ctrl + c
Uncomment	Ctrl + k, Ctrl + u
Formatting	Ctrl + k, Ctrl + d

Project Types

- **Windows app, Class Library**
  - A class library defines types and methods that are called by an application.
  - If the library targets .NET standard 2.0, it can be called by any .NET implementations that supports .NET standard 2.0
  - We can use a class library in our app or project by passing reference into it and then in the file using its namespace
  - Windows app is an app that can be run only on windows platform and can be executed by the operating system on the system directly.
  - Windows app in installed in windows platform using windows operating system.
  - Its interface is shown in the form of windows form.

- **Web application**
  - A consistent web development environment that is also used for creating the other components of the application.
  - Availability, performance, and scalability of web processing.
  - Access to the integrated .NET security model.

Create first C# program "Hello World"

- **What is namespace?**
  - Namespaces are used to logically arrange classes, structs, interfaces, enums, and delegates.
  - The namespace in c# can be nested.
  - The .NET framework already contains number of standard namespaces like System, System.Net, System.IO, etc.

- **Declaring a namespace**
  - The C3 language provide a keyword namespace to create a user defined namespace. The general form of declaring the namespace is as follows.

```
namespace <namespace_name>
{
    // Classes or interface or struct etc.
}
```

  - It is not possible to use any access specifiers like private, public, etc. with a namespace declaration. The namespace in C# are implicitly have public access and this is not modifiable.
  - The namespace elements can't be explicitly declared as private or protected. The namespace allows only public and internal elements as it members. The default is internal.

- **What is class?**
  - A class is like a blueprint of a specific object. In the real world, every object has some color, shape and functionalities.
  - In C#, class can be defined by using the class keyword.

```
public class <class name>
{
    // Fields, methods, properties, etc.
}
```

- **Variable and method declaration**

1. Method :
   - A method is a group of statements that together perform a task. Every C# program has at least one class with a method named Main.
   - The Main method is the entry point for every C# application and it's called by the common language runtime when the program is started.

```
<Access specifier> <Return type> <Method name> (parameter list)
{
    // Method body
}
```

Following are the various elements of the method –

- Access specifier – This determines the visibility of a variable or a method from another class.
- Return type – a method may return a value. The return type is the data type of the value of method returns.
- Method name – method name is unique identifier and it is case sensitive.
- Parameter list – parameters are optional, enclosed between parenthesis, the parameters are used to pass and receiver data from a method/
- Method body – this contains the set of instructions needed to complete the required activity.

2. Variable :
   - C# variable contains a data value of the specific data type.
   - Variable declaration in C# is as follows –

     [Modifier] [Data type] [Variable name] = value;

   - Variable name must be unique.
   - Variable name can contain letters, digits, and the underscore (_) only.
   - Variable name must start with a letter.
   - Variable names are case-sensitive, num and Num are considered different names.
   - Variable names cannot contain reserved keywords. Must prefix @ before keyword if want reserve keywords as identifiers.

Understanding C# Program

- **Program Flow**
  - C# program consists of the following parts –
    1. Namespace declarations
    2. A class
    3. Class methods
    4. Class attributes
    5. A main method
    6. Statements and expressions
    7. Comments
  - C# is case sensitive.
  - All statements and expression must end with a semicolon (;).
  - The program execution starts at the main method.
  - Unlike java, program file names could be different from class name.

- **Understanding syntax**

```
Using System;

Namespace HelloWorldApplication
{
    Class HelloWorld
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World");
        }
```

```
    }
}
```

- The first line of the program using System; - the using keyword is used to include the System namespace in the program. A program generally has multiple using statements.
- The next line has the namespace declaration. A namespace is a collection of classes. The HelloWorldApplication namespace contains the class HelloWorld.
- The next line has a class declaration, the class HelloWorld contains the data and method definitions that your program uses. Classes generally contain multiple methods. Methods define the behavior of the class. However, the HelloWorld class has only one method Main.
- The next line defines the Main method, which is the entry point for all C# programs. The Main method states what the class does when executed.
- The Main method specifies its behavior with the statement Console.WriteLine("Hello World");
- WriteLine is a method of the Console class defined in the System namespace. This statement causes the message "Hello World" to be displayed on the screen.

Working with the code files, Projects and solutions

- **Understanding structure of solution**
  - Solution is collections of projects which are related each other and are containing information of dependencies between them.
  - A solution can contain multiple projects and a project can be part of multiple solutions.
  - A solution contains projects, projects contain source files and solution also contains solutionName.sln file.

- **Understanding structure of project**
  - Structure of windows forms app file name is same as app name and [appname].csproj file contains references of other projects and of packages used and also version of project etc.
  - It has got dependencies which has all the server side NuGet Packages and other third-party packages required in the project.
  - It has a .cs file which contains code of windows form that we create.
  - It also contains Program.cs file which has Main method as required to run app as console application. The configuration for app is in CreateHostBuilder Method called in Main method.

- **Structure of web app**
  - File name is same as app name and [appname].csproj file contains references of other projects and of packages used and also version of project etc.
  - It contains connected services folder which is used to connect project to services like Azure and is basically used during deployment.

- It has got dependencies which has all the server side NuGet packages and other third party packages required in the project.
- Then there is properties folder which contains launchSettings.json which contains debug settings.
- Then it contains pages folder which contains some demo interface pages.
- wwwroot folder contains static files like's images, css, JavaScript, etc. These are the only files which are served over http request.
- It also contains Program.cs file which has Main method as required to run app as console application.
- It also contains Startup.cs file which runs always first when the project is executed and it contains appSettings.json is an application configuration file and contains configurations like database settings, any global variables for whole application.

- **Structure of class library**
  - File name is same as app name and [appname].csproj file contains reference of other projects and of packages used and also version of project etc.
  - It has got dependencies which has all the server side NuGet packages and other third party packages required in the project.
  - It has a .cs file which contains code of particular class that we define in it.

- **File extensions**
  - .sln – It contains information of the projects contained in the solution as it is the solution file.
  - .csproj – It contains the reference of other projects and of packages used and also version of project etc.
  - .cs – Class file of C#.
  - .json – JavaScript Object Notation file which stores simple objects and data structures.

Data types and Variables with Conversion

- C# is strongly-typed language.
- It means we must declare the type of a variable that indicates the kind of values it is going to store, such as integer, float, decimal, text, etc.

- **Value Data type:**
  - Assigns a value directly in both signed / unsigned form and the system allocates memory to store the value.
  - They are derived from the class System.
  - The value types directly contain data which stores numbers, alphabets, and floating-point numbers etc.

    1. Pre-defined data type:
       These are the already defined data types in C#
       ➔ Integer – int
       ➔ Decimal – decimal
       ➔ Float – float
       ➔ Character – char
       ➔ Double – double
    2. User defined data type:
       It is a data type which is defined and used by the users.
       ➔ Structure – struct
       ➔ Enumerations – enum

- **Reference Data type:**
  - The reference data types do not contain the actual data stored in a variable, but they contain a reference to the variables.
  - In other words, they refer to a memory locations.

- If the data in the memory location is changed by one of the variables, the other variable automatically reflects this change in value.

1. Predefined data type:
   ➔ Such as Objects, string.
2. User defined data type:
   ➔ Such as Classes, Interface

- **Data type Conversion**
  - **Implicit Conversion** :- No special syntax is required because the conversion always succeeds and no data will be lost. Examples include conversions from smaller to larger integral types, and conversions from derived classes to base classes.
  - **Explicit Conversion** :- Explicit conversions require a cast expression. Casting is required when information might get lost in the conversion, or when the conversion might not succeed for other reasons. Typical examples include numeric conversion to a type that has less precision or a smaller range, and conversion of a base-class instance to a derived class.
  - **User defined conversions** :- User-defined conversions are performed by special methods that you can defined to enable explicit and implicit conversions between custom types that do not have a class-derived class relationship. For more information, see user-defined conversion operators.

- **Boxing and Unboxing :-**
    - Boxing – Boxing is the process of converting a value type to the object type or any interface type implemented by this value type. Boxing is implicit.

      Example :-

      ```
      int numer1 = 100;
      long number2 = number1;
      ```

    - Unboxing – It is the process of converting a reference type to value type. Unboxing the value from the reference and assign it to a value type. Unboxing is explicit.

      Example :-

      ```
      string str = “45”;
      int number = Convert.ToInt32(str);
      int number1 = int.Parse(str);
      ```

Operators and Expressions

- C# provides a number of operators. Many of them are supported by the built-in types and allow you to perform basic operations with values of those types. Those operators include the following groups:

- Arithmetic operators that perform arithmetic operations with numeric operands
    - Arithmetic Operators - +, -, /, *, %, ++(Incremental), -- (Decremental) etc.
- Assignment Operators – They are used for assigning a value to a variable.
    - Assignment Operators - =, +=, -=, *=, /=, etc.
- Boolean logical operators that perform logical operations with bool operands.
    - Logical Operators - &&, ||, !
- Bitwise and shift operators that perform bitwise or shift operations with operands of the integral types.
    - Bitwise Operators - &, |, ^, >>, <<, etc.
- Relational Operators They are used to compare operands to obtain a result.
    - Relational Operators - ==, !=, <, >, >=, <=, etc.
- Misc Operators –They perform miscellaneous tasks like giving type of operand, size of operand etc.
    - Misc Operators – size of(), type of(), ?:(Ternary), etc.

- **Expressions**
  - Expressions are used to manipulate data. Like in Mathematics, expressions in programming languages, including C# are constructed from the operands and operators.

    Example :- simpleInterest = principal * time * rate / 100;

  - Expressions in C# comprise one or more operators that performs some operations on variables.
  - An operations is an action performed on single or multiple values stored in variables in order to modify them or to generate a new value with the help of minimum one symbol and value.

Understanding Decision making & statements

- If Statement: The if statement contains a boolean condition followed by a single or multi-line code block to be executed. At runtime, if a Boolean condition evaluates to true, then the code block will be executed, otherwise not.

  ```
  if(condition)
  {
      // code block to be executed when if condition evaluates to true
  }
  ```

- else if Statement: Multiple else if statements can be used after an if statement. It will only be executed when the if condition evaluates to false. So, either if or one of the else if statements can be executed, but not both.

  ```
  if(condition1)
  {
      // code block to be executed when if condition1 evaluates to true
  }
  else if(condition2)
  {
  // code block to be executed when
  // condition1 evaluates to false
  //condition2 evaluates to true
  }
  ```

- else Statement: The else statement can come only after if or else if statement and can be used only once in the if-else statements. The else statement cannot contain any condition and will be executed when all of the previous if and else if conditions evaluate to false.

  ```
  if(condition)
  {
      // code block to be executed when if condition evaluates to true
  }
  else
  {
      // code block will executed when id condition evaluates to false
  }
  ```

- switch Statement: The switch statement can be used instead of if else statement when you want to test a variable against multiple conditions. The switch statement starts with the switch keyword that contains a match expression or a variable in the bracket switch (match expression). The result of this match expression or a variable will be tested against conditions specified as cases, inside the curly braces { }. Each case includes one or more statements to be executed. The case will be executed if a constant value and the value of a match expression/variable are equal. The switch statement also contains a default label. The default label will be executed if no cases executed. break keyword is used to exit the program control from a switch case.

Example:

```
switch(match expression/variable)
{
case constant-value:
        statement(s) to be executed;
        break;
default:
        statement(s) to be executed;
        break;
}
```

Loop Iteration

The following statements repeatedly execute a statement or a block of statements:

- for statement: executes its body while a specified Boolean expression evaluates to true.

  Syntax -

  ```
  for (statement1; statement2; statement3;)
  {
      //code to be executed
  }
  ```

- The foreach statement: enumerates the elements of a collection and executes its body for each element of the collection.

  Syntax -

  ```
  foreach (type variablename in arrayname)
  {
      //code to be executed
  }
  ```

- The do statement: conditionally executes its body one or more times.

  Syntax -

  ```
  do
  {
      //code to be executed
  } while(condition)
  ```

- The while statement: conditionally executes its body zero or more times.

  Syntax -

  while(condition)

  {

  // code to be executed

  }

- break: It is used to jump out of a loop when a particular condition is met.
- Continue: It is used to continue the loop but breaking that particular iteration when continue statement is executed

Understanding Arrays

- You can store multiple variables of the same type in an array data structure. You declare an array by specifying the type of its elements.
- An array can be single-dimensional, multidimensional or jagged.
- The number of dimensions and the length of each dimension are established when the array instance is created. These values can't be changed during the lifetime of the instance.
- The default values of numeric array elements are set to zero, and reference elements are set to null.
- A jagged array is an array of arrays, and therefore its elements are reference types and are initialized to null.
- Arrays are zero indexed: an array with n elements is indexed from 0 to n-1.
- Array types are reference types derived from the abstract base type Array. All arrays implement IList, and IEnumerable. You can use the foreach statement to iterate through an array. Single-dimensional arrays also implement IList<T> and IEnumerable<T>.

  Syntax:

  < Typeof array > [ ] < arrayName >;

- Here first we must specify the data type along with the brackets [].
- After that we must specify the Name of the Array.

Defining and Calling Methods

- A method is a code block that contains a series of statements.
- A program causes the statements to be executed by calling the method and specifying any required method arguments.
- In C#, every executed instruction is performed in the context of a method.
- The Main method is the entry point for every C# application and it's called by the common language runtime.

Syntax:

< Access Specifier > < Return Type > < Method Type > (<Parameter List>)

{

    // method body

}

- Access Specifier – describes the access of the method to classes in the program expublic, private, protected, Internal.
- Return Type – can be a type of value which a method returns or it also can be void if a method returns nothing.
- Method name – method receives a user defined name.
- Parameter list – it is the list of parameters passed on as argument when it is called.
- Different type of parameters in method
    - Value type – It is the normal C# value parameter which means value is directly passed.
    - Ref. Type – References of a variable is passed as an argument which are assigned first. Argument passed consists of ‘ref’ keyword first. Any changes made in this argument in method will reflect on the variable in the calling method.

- Optional or type – These are the type of arguments which may be passed only if operations on them are required otherwise the function consists default values of these parameters. This parameter should only be passed after required parameters

Object Oriented Concepts

- **Static Constructor**
  - A static constructor is used to initialize static variable of the class and to perform a particular action only once.
  - Static constructor is called only once, no matter how many objects you create.
  - Static constructor is called before instance (default or parameterized) constructor.
  - A static constructor does not take any parameters and does not use any access modifiers.

- **Key Points of static constructor**
  - Only one static constructor can be created in the class.
  - It is called automatically before the first instance of the class created.
  - We cannot call static constructor directly. CLR

- **Copy Constructor**
  - The constructor which creates an object by copying variables from another object is called a copy constructor. The purpose of a copy constructor is to initialize a new instance of the values of an existing instance.
  - In C#, copy constructor is a parameterized constructor which contains a parameter of same class type. The copy constructor in C# is useful whenever we want to initialize a new instance of the values of an existing instance.
  - In simple words, we can say copy constructor is a constructor which copies a data of one object into another object.

- **Private Constructor**
  - When a constructor is created with private specifies, it is not possible for another classes to derive from this class, neither is it possible to create an instance of this class. They are usually used in classes that contain static members only. Some key points of a private constructor are:
    - One use of a private constructor is when we have only static members.
    - Once we provide a constructor that is either private or public or any, the compiler will not add the parameter-less public constructor to the class.
    - In the presence of parameter less private constructor you cannot create a default constructor.
    - We cannot inherit the class in which we have a private constructor.
    - We can have parameters in private constructor.

- **Static class in C#**
  - Classes that cannot be instantiated or inherited are known as classes and the static keyword is used before the class name that consists of static data members and static methods.
  - It is not possible to create an instance of a static class using the new keyword. The main feature of static classes are as follows:
    - They can only contain static members.
    - They cannot be instantiated or inherited and cannot contain instance constructors. However, the developer can create static constructors to initialize the static members.

- **Destructors in C#**
  - A destructor is a special method which has the same name as the class but starts with the character ~ before the class name and immediately de-allocates memory of objects that are no longer required.
  - Following are the features of destructors:
    - Destructors cannot be overloaded or inherited.
    - Destructors cannot be explicitly invoked.
    - Destructors cannot specify access modifiers and cannot take parameters.

- **Inheritance in C#**
  - The similarity in physical features of a child to that of its parent is due to the child having inherited these features from its parents.
  - Similarly, in C#, inheritance allows you to create a class by deriving the common attributes and methods of an existing class.
  - Inheritance provides reusability us to extend an existing class.
  - The reason behind OOP Programming is to promote the reusability of code and to reduce complexity in code and it is possible by using inheritance.
  - The inheritance concept is based on a base class and derived class. Let us see the definition of a base and derived class.
    - Base Class – is the class from which features are to be inherited into another class.
    - Derived Class – it is the class in which the base class features are inherited.

- **Single Inheritance**
  - It is the type of inheritance in which there is one base class and one derived class.

- **Hierarchical Inheritance**
  - This is the type of inheritance in which there are multiple classes derived from one base class.
  - This type of inheritance is used when there is a requirement of one class feature that is needed in multiple classes.

- **Multi-Level Inheritance**
  - When one class is derived from another derived class then this type of inheritance is called multilevel inheritance.

- **Constructor in inheritance**
  - A constructor is a method with the same name as the class name and is invoked automatically when a new instance of a class is created.
  - Constructors of both classes must be executed when the object of child class is created.
  - Sub class's constructor invoked constructor of super class.
  - Explicit call to the super class constructor from sub class's can be made using base().
  - base() should be the first statement of child class constructor.
  - If u don't write base() explicitly then java compiler implicitly write the base().

- **Access modifiers**

- C# provides you with access modifiers that allow you to specify which classes can access the data members of a particular class.
- In C#, there are four commonly used access modifiers
    - public
    - private
    - protected
    - internal

	Applicable to the Application	Applicable to current class	Applicable to the derived class	Applicable to outside the namespace / assembly	Applicable to outside the namespace but in derived class
public	Yes	Yes	Yes	Yes	Yes
Private	No	Yes	No	No	No
protected	No	Yes	Yes	No	Yes
internal	Yes	Yes	Yes	No	No

- **Rules for access modifiers**
    - Members of same class can access each other.
    - Only public and internal access modifiers are used with the class.

- **Access Modifiers**
    - PUBLIC – The public access modifier provides the most permissive access level.
    - The members declared as public can be accessed anywhere in the class as well as from other classes.

- PRIVATE – The private access modifier provides the least permissive access level.
- Private members are accessible only within the class in which they are declared.
- PROTECTED – The protected access modifier allows the class members to be accessible within the class as well as within the derived classes.
- INTERNAL – The internal access modifier allows the class members to be accessible only within the classes of the same namespace / assembly.
- An assembly is a file that is automatically generated by the complier upon successful compilation of a .NET application.
- The code declares a variable called NumOne as internal, which means it has only assembly-level-access.

- **Encapsulation**
  - Encapsulation is one of the four fundamental OOP concepts. The other three are inheritance, polymorphism, and abstraction.
  - Encapsulation in C# is a mechanism of wrapping the data and code acting on the data together as a single unit.
  - In encapsulation, the variable of a class will be hidden from other classes, and can be accessed only through the methods or properties of their current class. Therefore, it is also known as data hiding.
  - In a different way, encapsulation is a protective shield that prevents the data from being accessed by the code outside the shield.
  - Encapsulation is the procedure of encapsulating data and functions into a single unit (called class).
  - To achieve encapsulation in C#-

    - Declare the variable of a class as private.
    - Provide public setter and getter methods or properties to modify and view the variables values.
  - The fields of a class can be made read-only and write-only.
  - A class can have total control over what is stored in its fields.

- **Why do we need encapsulation?**
  - The need of encapsulation is to protect or prevent the code from accidental corruption due to silly little errors that we are all prone to make.

- **What are properties in C#**
  - Properties allow you to control the accessibility of a class variables, and are the recommended way to access variables from the outside in C#.
  - A property is much like a combination of a variable and a method – it can't take any parameters, but you are able to process the value before it's assigned to our returned.
  - Properties are like data fields (variables), but have logic behind them.
  - From the outside, they look like any other member variable, but they act like a member function.
  - Defined like a field, with "GET" and "SET" accessors code added.
  - Properties are also used for encapsulation.
  - Types of properties in C#
    - Read/ write properties
    - Read only properties
    - Write properties
    - Auto implemented properties

- **Static property in C#**
  - The static property is used to access and manipulate static fields of a class in a safe manner.
  - The static property declared by using the static keyword.
  - The static property accessed using the class name and thus, belongs to the class rather than just an instance of the class.
  - The static property called by a programmer without creating an instance of the class.
  - We cannot initialize instance fields within static property.

- **Polymorphism in C#**
  - Polymorphism is one of the four pillars of Object Oriented Programming.
  - Polymorphism in C# is a concept by which we can perform a single action by different ways.
  - Polymorphism is derived from 2 Greek words: POLY and MORPHS.
  - The word "poly" means many and "morphs" means forms.
  - So polymorphism means many forms.
  - There are two types of polymorphism
    - Static polymorphism (Compile time polymorphism)
    - Dynamic polymorphism (Run time polymorphism)

- **Static polymorphism (Compile time polymorphism) in C#**
  - The mechanism of linking a function with an object during compile time is called static polymorphism or early binding.
  - It is also called static binding.

  C# provides two techniques to implement static polymorphism. They are

  - Method or function overloading

- Operator overloading

- **Method or function overloading**
  - You can have multiple definitions for the same function name in the same scope.
  - The definition of the function must differ from each other by the types and/ or the number of arguments in the argument list.
  - You cannot overload function declarations that differ only by return type.

- **Operator Overloading**
  - The concept of overloading a function can also be applied to operators.
  - Operator overloading gives the ability to use the same operator to do various operations.
  - It provides additional capabilities to C# operators when they are applied to user-defined data types.
  - It enables to make user-defined implementations of various operations where one or both of the operands are of a user-defined class.
  - Only the predefined set of C# operators can be overloaded.
  - To make operations on a user-defined data type is not as simple as the operations on a built-in-data type.
  - To use operators with user-defined data types, they need to be overloaded according to a programmer’s requirement.
  - An operator can be overloaded by defining a function to it.
  - The function of the operator is declared by using the operator keyword.
  - Operators may be considered as functions internal to the compiler.

Operators	Description
+, -, !, ~, ++, --	Unary operators take one operand and can be overloaded.
+, -, *, /, %	Binary operators take two operands and can be overloaded.
==, !=, =	Comparison operators can be overloaded.
&&, ||	Conditional logical operators cannot be overloaded directly.
+=, -=, *=, /=, %=, =	Assignment operators cannot be overloaded.

- **Method Hiding in C#**
  - Method hiding occurs in inheritance relationship when base class and derived class both have a method with same name and same signature.
  - When we create the object of derived class it will hide the base class method and will call its own method and this is called method hiding or name hiding in C# inheritance.
  - We use "new" keyword in derived function name to show that implementation of the function in derived class is intentional and derived class no longer want to use base class method.

  NOTE :- If we do not use "new" keyword then complier will raise only warning, but, program will work fine.

  DIFFERENT WAYS TO CALL A HIDDEN BASE CLASS MEMBER FROM DERIVED CLASS

  1. USE BASE KEYWORD
  2. CAST CHILD TYPE TO PRENT TYPE AND INVOKE THE HIDDEN MEMBER.
  3. PARENTCLASS PC = new Childclass();

- PARENT CLASS CAN HAVE THE REFERENCE OF ITS CHILD CLASS.
- WHEN WE CREATE THE OBJECT OF CHILD CLASS, PARENT CLASS OBJECT IS ALSO CREATED.

- **DYNAMIC OR RUNTIME POLYMORPHISM IN C#**
  - RUN TIME POLYMORPISM IS ACHIEVED BY METHOD OVERRIDING.
  - METHOD OVERRIDING ALLOWS US TO HAVE VIRTUAL AND ABSTRACT METHODS IN THE BASE USING DERIVED CLASSES WITH THE SAME NAME AND THE SAME PARAMETERS.

- **C# method overriding**
  - If derived class defines same method as defined in its base class, it is known as method overriding in C#.
  - It is used to achieve runtime polymorphism.
  - It enables you to provide specific implementation of the class method in child class which is already provided by its base class.
  - To perform method overriding in C#, you need to user virtual keyword with base class method and override keyword with derived class method.
  - A method declared using the virtual keyword is referred to as virtual method.
  - In the derived class, you need to declare the inherited virtual method using the override keyword.
  - In the derived class, you need to declare the inherited virtual method using the override keyword which is mandatory for any virtual method that is inherited in the derived class.
  - The override keyword overrides the base class method in the derived class.

- **Abstraction**
  - Abstraction is one of the principle of object oriented programming. It is used to display only necessary and essential features of an outside the world. Means displaying what is necessary and encapsulate the unnecessary things to outside the world. Hiding can be achieved by using "private" access modifier.
  - Abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user.
  - In other words, the user will have the information on **what** the object does instead of **how** it does it.

    Note – Outside the world means when we use reference of object then it will show only necessary methods and properties **(Abstraction)** and hide methods which are not necessary **(Encapsulation).**

- **Implementation of abstraction**
  - To implement abstraction let's take an example of a car. We knows a car, Car is made of name of car, color of car, steering, gear, rear view mirror, brakes, silencer, exhaust system, diesel engine, car engine, and other internal machine details etc.
  - Now let's think in terms of car rider or a person who is riding a car. So to drive a car what a car rider should know from above category before he starts a car driving.
  - Necessary things means compulsory to know before starting a car
    - Name of car
    - Color of car
    - Steering
    - Rear view mirror
    - Brakes

    - Gear
- Unnecessary things means not that compulsory to know for a rider
    - Internal details of car
    - Car engine
    - Diesel engine
    - Exhaust system
    - Silencer
- Abstraction can be achieved using abstract classes, abstract methods and interface as well.

- **Abstract class**
    - The word abstract means a concept or an idea not associated with any specific instance. In programming we apply the same meaning of abstraction by making classes not associated with any specific instance.
    - The abstraction is done when we need to only inherit from a certain class, but not need to instantiate objects of that class.
    - In C#, abstraction is achieved using abstract classes and interfaces.
    - Java abstract classes are used to declare common characteristics of subclasses.
    - A class which contains the abstract keyword in its declaration is known as abstract class.
    - It can only be used as a BASE class for other classes that extend the abstract class.
    - Abstract classes may or may not contain abstract methods, i.e., methods without body (public void get();)
    - Like any other class, an abstract class can contain fields that describe the characteristics and methods that describe the actions that a class can perform.

- But, is a class has at least one abstract method, then the class must be declared abstract.
- If a class is declared abstract, it cannot be instantiated.
- To use an abstract class, you have to inherit it from another class, provide implementations to the abstract methods in it.
- If you inherit an abstract class, you have to provide implementations to all the abstract methods in it.
- An abstract class can implement code with non-abstract methods.
- An abstract class can have modifiers for methods, properties, etc.
- An abstract class can have constants and fields.
- An abstract class can implement a property.
- An abstract class can have constructors or destructors.

- **Abstract properties**
  - The word abstract means incomplete, which means no implementation in programming.
  - This is a duty of child to implement an abstract member of their parent class.
  - Declared by using the abstract keyword.
  - Contain only the declaration of the property without the bod of the get and set accessors (which do not contain any statements and can be implemented in the derived class).
  - Are only allowed in an abstract class.

- **Interface**
  - An interface is a contract between itself and any class that implements it. This contract states that any class that implements the interface will implement the interface's properties, methods and/or events. An interface contains no implementation, only the

signature of the functionality the interface provides. An interface can contain signature of methods, properties, indexers & events.

- The interface defines that **'what'** part of the syntactical contract and the deriving classes define the **'how'** part of the syntactical contract.
- Interface define properties, methods, and events, which are the members of the interface. Interfaces contain only the declaration of the members. It is the responsibility of the deriving class to define the members.
- One of the main reason to introduce interfaces is that it can be used in multiple inheritance.
    - An interface contains only abstract members, just like classes interface also contains properties, methods, delegates or events, but only declarations, no implementations.
    - An interface cannot be instantiated but can only be inherited by classes or other interfaces.
    - Interface cannot have fields.
    - An interface is declared using the keyword interface.
    - In C#, by default, all members declared in an interface have public as the access modifier. They don't allow explicit modifiers.

- **Implementing interface**
    - An interface is implemented by a class in a way similar to inhering the class.
    - When implementing an interface in a class, implement all the abstract methods declared in the interface. If all the methods are not implemented, the class cannot be compiled.

- The methods implemented in the class should be declared with the same name and signature as defined in the interface.

- **Interface inheritance**
  - Interface can inherit from other interfaces.
  - A class that inherits this interface must provide implementation for all interface members in the entire interface inheritance chain.
  - Interface reference variable can have the reference of their child class.

- **Explicit interface implementation**
  - There are 2 types of implementation of interfaces.
    - Implicit interface implementation
    - Explicit interface implementation
  - A class has to explicitly implement multiple interfaces if these interfaces have methods with identical names.
  - If an interface has a method name identical to the name of a method declared in the inheriting class, this interface has to be explicitly implemented.

- **Similarities between abstract class and interfaces**
  - Abstract classes and interfaces both declare methods without implementing them.
  - Both, abstract classes as well as interfaces, contain abstract methods.
  - Abstract methods of both, the abstract class as well as the interface, are implemented by the inheriting subclass.
  - Both, abstract classes as well as interfaces, can inherit multiple interfaces.

- **Multiple Inheritance**
  - Multiple inheritance is a feature in which a class can inherit characteristics and features from more than one parent class.
  - A class can extends or inherits only one class but a class can implements or inherits one or more than one interface.
  - In other words, a sub class can only have one parent class but a subclass can implements or inherits one or more than one interfaces.

- **Collections in C# Programming**
  - A collection is a set of related data that may not necessarily belong to the same data type that can be set or modified dynamically at run-time.
  - In other words, collection is a dynamic array.
    - Its length can increase on runtime.
  - Normal array's length is fixed, it means we cannot change the length after declaring an array.
  - Array.Resize()
  - Resize method of an array destroys old array and create a new array with new length.
  - We can never insert a value into a middle of an array, because if we want to do this then array length should be increased but we cannot increase the length of an array after declaring the length of an array.
  - We can never delete a value from a middle of an array.
  - Accessing collections is similar to accessing arrays, where elements are accessed by their index numbers. However, there are differences between arrays and collections in C#.

Array	Collection
Cannot be resized at run-time.	Can be resized at run-time.
The individual elements are of the same data type.	The individual elements can be of different data types.
Do not contain any methods for operations on elements.	Contain methods for operations on elements.
We can never insert a value into a middle of an array.	We can insert a value into a middle of a collection.
We can never delete a value from a middle of an array.	We can delete a value from a middle of a collection

- Collections were introduced in C# 1.0
- We have two kinds of collections.
    - Non-generic collections
        - Stack, ArrayList, HashTable, SortedList etc.
        - System.Collections namespace have non-generic collections.
    - Generic collections
        - List<T>, LinkedList<T>, Queue<T>, SortedList<T, V>.
        - System.Collections.Generic namespace have generic collections.
- Collections have a mechanism called auto-resizing.
- Capacity property which tells number of items that can be stored under any collection.
- Insert method.
- Remove and RemoveAt method.
- All non-generic collection classes have three features.
    - They can store homogeneous elements as well as heterogeneous elements.

    - They are not type safe.
    - It can automatically resize dynamically (Auto Resizing).

- **Stack Non-Generic Collection**
  - C# includes a special type of collections which stores elements in LIFO style (Last In First Out).
  - Stack allows null value and also duplicate values. It provides a Push() method to add a value and Pop() or Peek() methods to retrieve values.
  - A stack is used to create a dynamic collection which grows, according to the need of your program.
  - The capacity of a stack is the number of elements the stack can hold. As elements are added to a stack, the capacity is automatically increased as required through application.
  - In a stack, you can store elements of the dame type or different types.
  - The stack class implements the IEnumerable, ICollection, and ICloneable interface.
  - When you add an item in the list, it is called pushing the element.
  - When you remove it, it is called popping the element.

- **Important properties and methods of stack**
  - Count – Returns the total count of elements in the stack.
  - Push – Inserts an item at the top of the stack.
  - Peek – Returns the top item from the stack.
  - Pop – Removes and returns items from the top of the stack.
  - Contains – Checks whether an item exists in the stack or not.
  - Clear – Removes all items from the stack.

- **Queue Non-Generic Collection**
  - C# includes a Queue collection class in the System.Collection namespace.
  - Queue stores the elements in FIFO Style (First In First Out), exactly opposite of the stack collection.
  - It contains the elements in the order they were added.
  - Queue collection allows multiple null and duplicate values.
  - Use the Enqueue() method to add values and the Dequeue() retrieve the values from the Queue.

- **Properties and methods of Queue**
  - Count – Returns the total count of elements in the Queue.
  - Enqueue – Adds an item into the queue.
  - Dequeue – Removes and returns an item from the beginning of the queue.
  - Peek – Returns first item from the queue.
  - Contains – Checks whether an item is in the queue or not.
  - Clear – Removes all the items from the queue.

- **Hash Table collection in C#**
  - HashTable stores data in key/value format.
  - Array and ArrayList also stores data in key/value format.
  - In Array or ArrayList keys are pre-defined i-e index numbers, it means you cannot explicitly define keys in Array or ArrayList.
  - In HashTable, keys are not pre-defined it means you can explicitly define user-defined keys in HashTable.
  - C# includes HashTable collection in System.Collections namespace.
  - The HashTable collection stores key-value pairs.

- The HashTable class represents a collection of key-and-value-pairs that are organized based on the hashcode of the key.
- It uses the key to access the elements in the collection.
- A hash table is used when you need to access elements by using the key, and you can identify a useful key value.
- Each item in the hash table has a key/value pair. The key is used to access the items in the collection.

- **Hash-Code**
  - Hash tables used in hashing algorithm to generate hash codes for every key and because of these hash codes hash tables are faster than array and arraylist.

- **Methods and Properties of Hash Table**
  - Add – Adds an item with a key and value into the hashtable.
  - Remove – Removes the item with the specified key from the hashtable.
  - Clear – Removes all the items from the hashtable.
  - Contains – Checks whether the hashtable contains a specific key.
  - ContainsKey – Checks whether the hashtable contains a specific key.
  - ContainsValue – Checks whether the hashtable contains a specific value.
  - GetHashCode – Returns the hashcode for the specified key.
  - Count – Gets the total count of key/value pairs in the Hashtable.
  - Keys – Gets an ICollection of keys in the Hashtable.
  - Values – Gets an ICollection of values in the Hashtable.

- **Difference between abstract classes and interface**

Abstract class	Interface
An abstract class can inherit a class and multiple interfaces.	An interface can inherit multiple interfaces but cannot inherit a class.
An abstract class can have methods with a body.	An interface cannot have methods with a body.
An abstract class method is implemented using the override keyword.	An interface method is implemented without using the override keyword.
An abstract class is a better option when you need to implement common methods and declare common abstract methods.	An interface is a better option when you need to declare only abstract methods.
An abstract class can declare constructors and destructors.	An interface cannot declare constructors and destructors.
If you want to declare an abstract member in abstract class then abstract keyword is mandatory.	If you want to declare an abstract member in interface then abstract keyword is not mandatory because members of inheritance are abstract by default.

- **Difference between non-generic collection and generic collection**

Non-generic collection	Generic collection
These are the collection that can hold elements of different data types.	These are the collection that can hold data of same type.
It hold elements as Object type. So it included overhead of type conversions.	It reduced overhead of type-conversions.
These are also called loosely typed.	These are also called strongly typed.
Not type safe.	Type safe.
Not secure.	Secure

- **List<T> Generic collection**
  - A list is one of the generic collection classes in the "SystemCollection.Generic" namespace.
  - There are several generic collection classes in the System.Collection.Generic namespace that includes the following:
    - List
    - Dictionary
    - Stack
    - Queue
  - An ArrayList resizes automatically as it grows(Auto- Resizing).
  - The List<T> collection is the same as ArrayList except that List<T> is a generic collection whereas ArrayList is a non-generic collection.
  - List<T> class can accept null as a valid value for reference types and it also allows duplicate elements.

- List<T> class is not sorted by default and elements are accessed by zero-based index.

- **Why to use a List**
  - Unlike arrays, a List can grow in size automatically in other words a list cam be re-sized dynamically but arrays cannot.
  - List is a generic type
  - List is type safe.
  - The list class also provides methods to search, sort and manipulate class.

- **Properties and methods of class**
  - Capacity – Gets or sets the total number of elements the internal data structure can hold.
  - Count – Gets the number of elements contained in the List<T>.
  - Add() – Adds an object to the end of the list.
  - AddRange() – Adds the elements of the specified collection to the end of the list.
  - Insert() – Inserts an element into the list at the specified index.
  - InsertRange() – Inserts the elements of a collection into the list at the specified index.
  - Remove() – Removes the first occurrence of a specific object from the list.
  - RemoveAt() – Removes the element at the specified index of the list.
  - RemoveRange() – Removes a range of elements from the list.
  - RemoveAll() – Removes all the elements that match the conditions defined by the specified predicate.

- IndexOf() – Returns the zero-based index of the first occurences of a value in the list.
- LastIndexOf() – Returns the zero-based index of the last occurrence of a value in the list.
- Clear() – Removes all elements from the list.
- Find() – Searches for an element that matched the conditions defined by the specified predicate, and returns the first occurrence within the entire list.
- FindLast() – Searches for an element that matched the conditions defined by the specified predicate, and returns the last occurrence within the entire list.
- FindAll() – Retrieves all the elements that match the conditions defined by the specified predicate.
- ToArray() – Copies the elements of the list to a new array.
- ToList() – Copies the elements of the array to a new list.

- **Null Coalesce ?? operator in C#**
  - The ?? operator is also known as the null-coalescing operator.
  - It returns the left side operand if the operand is not null else it returns the right side operand.
  - Coalescing operator returns the first non-null value from the chain.

- **C# Enum**
  - Enum is a set of constants.
  - An enum is a special ”class” that represents a group of constants (unchangeable / read-only variables)
  - Enum is short for “enumerations”, which means “specifically listed”.
  - To create an enum, use the enum keyword (instead of class or interface), and separate the enum items with a comma.

- In C#, an enum (or enumeration type) is used to assign constant names to a group of numeric integer values.
- It makes constant values more readable, for example, WeekDays.Monday is more readable then number 0 when referring to the day in a week.
- An enum is defined using the enum keyword, directly inside a namespace, class, or structure. All the constant names can be declared inside the curly brackets and separated by comma.
- The default underlying type of an enum is int.
- The default value for first element is ZERO and gets incremented by 1.
- Enums are value types.
- Enums are more readable and maintainable,
- Enum is converted into abstract class behind the scenes.
- If values are not assigned to enum members, then the compiler will assign integer values to each member starting with zero by default.
- The first member of an enum will be 0, and the value of each successive enum member is increased by 1.
- You can assign different values to enum member.
- A change in the default value of an enum member will automatically assign incremental values to the other members sequentially.

- **Sealed class in C#**
  - A sealed class is a class that prevents inheritance.
  - The features of a sealed class are as follows:
    - A sealed class can be declared by preceding the class keyword with the **sealed** keyword.
    - The sealed keyword prevents a class from being inherited by any other class.

    - The sealed class cannot be a base class as it cannot be inherited by any other class. If a class tries to derive a sealed class, the C# compiler generated error.
- Purpose of Sealed class
    - Consider a class named SystemInformation that consists of critical methods that affect the working of the operating system.
    - You might not want any third party to inherit the class SystemInformation and override its methods, thus, causing security and copyright issues.
    - Here, you can declare the SystemInformation class as sealed to prevent any change in its variables and methods.

- **Sealed Methods in C#**
    - When the derived class override a base class method, variable, property or event, then the new method, variable, property, or event can be declared as sealed.
    - Sealing the new method prevents the method from further overriding.
    - An overridden method can be sealed by preceding the override keyword with the sealed keyword.
    - Steps to remember for sealed methods
        - Sealed method is always override method of child class
        - We cannot again override the sealed method
        - Sealed method is only available with method overriding.
        - Sealed keyword is not available with the method hiding.
        - Sealed is used together with override keyword,
        - We cannot make normal methods as sealed.

- **Indexer in C#**
  - Indexers allow our object to be used just like an array, or we can say we can index the object using [] brackets just like arrays.
  - We can say indexers are special type of properties which adds logic that how can array or list store the values.
  - Syntax of indexer resembles to properties.
  - We can use all access modifiers with indexers and also have return types.
  - Indexers are the regular members of a class.
  - Indexers is created with the help of this keyword.
  - In C#, introduce new concept is indexer. This is very useful for some situation. Let as discuss something about indexer.
    - Indexer concept is object act as an array.
    - Indexer an object to be indexed in the same way as an array.
    - Indexer modifier can be private, public, protected, or internal.
    - The return type can be any valid C# types.
    - Indexers in C# must have at least one parameter. Else the compiler will generate a compilation error.

- **Delegates**
  - Delegate meaning from Google: A person sent or authorized to represent others, in particular an elected representative sent to a conference.
  - Delegate is a type which holds a method's reference in an object.
  - It is also called function pointer.
  - Delegate is of reference type.
  - Delegate signature should be as same as the method signature referencing by a delegate.

- Delegate can point to a parameterized method or non-parameterized method.
- Delegate has no implementation means no body with {}.
- We can use invoke() method with delegates.
- Delegates are used to encapsulate methods.
- In the .net framework, a delegate points to one or more methods. Once you instantiate the delegate, the corresponding methods invoke.
- Delegates are objects that contain references to methods that need to be invoked instead of containing the actual method names.
- Using delegates, you can call any method, which is identified only at run-time.
- In C#, invoking a delegate will execute the referenced method at run-time.
- To associative a delegate with a particular method, the method must have the same return type and parameter type as that of the delegate.

- **Single cast delegate**
  - Single cast delegate point to single method at a time. In this the delegate is assigned to a single method at a time. They are derived from System.Delegate class.

- **Multiple delegate**
  - In C#, a user can invoke multiple delegates within a single program.
  - Depending on the delegate name or the type of parameters passed to the delegate, the appropriate delegate is invoked.

- **Multi cast delegate**
  - When a delegate is wrapped with more than one method that is known as a multicast delegate.
  - In C#, delegates are multicast, which means that they can point to more than one function at a time. They are derived from System.MulticastDelegate class.
  - We can use += and -= assignment operators to implement multi cast delegates.

- **Anonymous Function**
  - We discussed that delegates are used to reference any methods that has the same signature as that of the delegate.
  - As the name suggests, an anonymous method is a method without a name just the body.
  - Anonymous methods in C# can be defined using the delegate keyword.
  - It is introduced in C# 2.0.
  - Anonymous method can be assigned to a variable of delegate type.
  - You need not specify the return type in an anonymous method; it is inferred from the return statement inside the method body.
  - We don’t required to use access modifiers with anonymous function like public, private etc.
  - We don’t required to use return type like int, string because its return type is set as same as delegate type.
  - Anonymous function is not a static and instance member.
  - Lesser typing work because we don’t required to write access modifier, return type and name of the function.
  - Anonymous functions are suggested when code volumes are less.
  - It cannot contain jump statement like goto, break and continue.

- It cannot access ref or out parameter of an outer method.
- Points to remember
    - Anonymous method can be defined using the delegate keyword.
    - Anonymous method must be assigned to delegate.
    - Anonymous method can access outer variables or functions.
    - Anonymous method can be passes as a parameter.
    - Anonymous method can be used as event handlers.

Ex:- saveButton.Click += delegate(Object o, EventArgs e)
{
    MessageBox.Show("Save Successfully!");
};

- **Lambda Expression**
    - C# 3.0 introduced the lambda expression.
    - It is also works like an anonymous method.
    - The difference that in lambda expressions you don't need to specify the type of the value that you input making it more flexible to use.
    - It means lambda expression simplifies the anonymous function or you can say that lambda expression is a shorthand for an anonymous function.
    - The lambda expression can be of two types:
        1. Statement lambda: - consists of the input and a set of statements to be executed.
           Syntax:
           Input => { statements };
           Note: Does not return any value implicitly.

2. Expression Lambda: Consists of the input and the expression.
   Syntax:
   Input => expression;
   Note: Returns the evaluated value implicitly.

- **Generics**
  - Generics introduced in C3 2.0.
  - Generics allow you to write a class or method that can work with any data type.
  - Generics allow you to define a class with placeholders for the type of its fields, methods, parameters, etc. Generics replace these placeholder with some specific type at compile time. It helps you to maximize code reuse, type safety, and performance.
  - You can create your own generic interfaces, classes, methods, events and delegates.
  - You may get information on the types used in a generic data type at run-time.
  - A generic class or method can be defined using angle brackets<>
  - Generics can be applied to the following.
    - Interface
    - Abstract class
    - Class
    - Method
    - Static method
    - Property
    - Event
    - Delegates
    - Operator
  - Advantages of Generic

    - Increase the reusability of code.
    - Generic has a performance advantage because it removes the possibilities of boxing and unboxing.

- **Generic Methods**
  - Generic methods process values whose data types are known only when accessing the variables that store these values.
  - A generic method is declared with the generic type parameter list enclosed within angular brackets.
  - Defining methods with type parameters allow you to call the method with a different type every time.
  - You can declare a generic method within generic or non-generic class declarations.
  - Generic methods can be declared with the following keywords.
    - Virtual
    - Override
    - Abstract

- **Generic Class**
  - Generic classes define functionalities that can be used for any data type and are declared with a class declaration followed by a type parameter enclosed within angular brackets.
  - Generic introduced in C# 2.0. Generics allow you to define a class with placeholder for the type of fields, methods, parameters, etc.
  - Generics replace these placeholders with some specific type at compile time.

- **Dictionary**
  - Now we want to create a collection in which we don't want to access elements through index.
  - We want to store data in key value format where keys are user-defined.
  - We want to insert same type of data in collection.
  - In C#, dictionary is a generic collection which is generally used to store key/value pairs.
  - The working of dictionary is quite similar to the non-generic hashtable.
  - The advantage of dictionary is, it is generic type.
  - Dictionary is defined under System.Collection.Generic namespace.
  - It is dynamic in nature means the size of the dictionary is grows according to the need.
  - The dictionary <Tkey, TValue> collection in C# is same as English dictionary.
  - English dictionary is a collection of words and their definitions, often listed alphabetically in one or more specific languages.
  - In the same way, the dictionary in C# is a collection of Keys and Values, where key is like word and value is like definition.
  - The Dictionary<TKey, TValue> class is generic collection class in the System.Collection.Generics namespace.
  - TKey denotes the key and TValue is the type of TValue

- **Important properties and methods of Dictionaries**
  - Count – Gets the total number of elements exists in the Dictionary<TKey, TValue>.
  - Keys – Returns collection of keys of dictionary.
  - Values – Returns collection of values of dictionary.

- Add – Adds an item to the dictionary collection, add key-value pairs in dictionary collection.
- Remove – Removes the element with the specified key.
- ContainsKey – Checks whether the specified key exists in dictionary.
- ContainsValue – Checks whether the specified value exists in dictionary.
- Clear - Removes all elements from dictionary.
- TryGetValue – Returns true and assigns the value with specified key, if key does not exists then return false.

- **Important points**
  - In dictionary, the key cannot be null, but value can be.
  - In dictionary, key must be unique. Duplicate keys are not allowed if you try to use duplicate key then compiler will throw an exception.
  - In dictionary, you can only store same types of elements.
  - The capacity of a dictionary is the number of elements that dictionary can hold.