➔ Describe the different C# classes in detail.

- There are 4 types of classes that we can use in C#:
  - **Static Class**: It is the type of class that cannot be instantiated, in other words, we cannot create an object of that class using the new keyword and the class members can be called directly using their class name.
  - **Abstract Class**: classes are declared using the abstract keyword. Objects cannot be created for abstract classes. If you want to use it then it must be inherited in a subclass.
  - **Partial Class**: is a type of class that allows dividing their properties, methods, and events into multiple source files, and at compile time these files are combined into a single class.
  - **Sealed Class**: are used to restrict the inheritance feature of object-oriented programming. Once a class is defined as a sealed class, the class cannot be inherited.

➔ An Abstract class is a class which is denoted by abstract keyword and can be used only as a Base class. This class should always be inherited. An instance of the class itself cannot be created. If we do not want any program to create an object of a class, then such classes can be made abstract.

- Any method in the abstract class does not have implementations in the same class. But they must be implemented in the child class.
  - Example:

```
abstract class AB1
{
Public void Add();
}
Class childClass : AB1
{
childClass cs = new childClass ();
int Sum = cs.Add();
}
```

- All the methods in an abstract class are implicitly virtual methods. Hence, the virtual keyword should not be used with any methods in the abstract class.

➔ Differences between public, static, and void:

- Public declared variables or methods are accessible anywhere in the application.
- Static declared variables or methods are globally accessible without creating an instance of the class.
  - Static member are by default not globally accessible it depends upon the type of access modified used (public, private, etc). The compiler stores the address of the method as the entry point and uses this information to begin execution before any objects are created.
- Void is a type modifier that states that the method or variable does not return any value.

➔ Access Modifiers are:

- public – there are no restrictions on accessing public members
- private – access is limited to within the class definition. This is the default access modifier type if none is formally specified
- protected – access is limited to within the class definition and any class that inherits from the class
- internal – access is limited exclusively to classes defined within the current project

assembly
- protected internal – access is limited to the current assembly and types derived from the containing class, all members in the current project and all members in the derived class can access the variables
- private protected – access is limited to the containing class or types derived from the containing class withing the current assembly

➔ Interface – is a class with no implementation, it contains the declaration of methods, properties, and events. By default, the members of the interface class are abstract and public with no fields defined. If you want to access the interface methods then the interface must be implemented by another class using ':' symbol. If you want to define the body of the methods that can only be implemented in the implementing class.

➔ Namespaces are designed for providing a way to keep one set of names separate from another. The class names declared in one namespace do not conflict with the same class names declared in another, helps control the scope of class and method names in larger programming projects.

➔ Collection classes in C#
- Collection classes are classes that are mainly used for data storage and retrieval, allocating dynamic memory during run time and access the items of the collection using the index value that makes the search easier and faster. These collection classes belong to the object class.
  - **Array lis**t: refers to the ordered collection of the objects that are indexed individually. You can use it as an alternative to the array. Using index you can easily add or remove the items off the list and it will resize itself automatically. It works well for dynamic memory allocation, adding or searching items in the list.
  - **Hash table**: use to access the item of the hash table- use the key-value to refer to the original assigned value to the variable. Each item in the hash table is stored as a key/value pair and the item is referenced with its key value.
  - **Stack**: works on the concept of last-in and first-out (LIFO) collection of the objects.
    ➔ Push - an item onto the list
    ➔ Pop - removes the item from the list
  - **Queue**: this collection works on the concept of first-in and first-out (FIFO) collection of the object.
    ➔ Enqueue - adding an item to the list
    ➔ Deque - removing the item from the list.
  - **Sorted lis**t: this collection class uses the combination of key and the index to access the item in a list.
  - **BitArray**: this collection class is used to represent the array in binary form (0 and 1), use this collection class when you do not know the number and the items can be accessed by using integer indexes that start from zero.

➔ Differences between Class and Struct:
- Class supports inheritance, is passed by reference, is private by default, good for larger complex objects, and uses waste collection for memory management
- Struct does not support inheritance, is passed by value, is public by default, good for small isolated models, does not use waste collection thereby does not have memory management, causes additional overhead, but faster retrival when stored on the stack

➔ A type is said to be nullable if it can be assigned a value or can be assigned null, which means the type has no value whatsoever. By default, all reference types, such as String, are nullable, but all value types, such as Int32, are not.
- For example, you can store any value from -2,147,483,648 to 2,147,483,647 or null in a Nullable<Int32> variable. Similarly, you can assign true, false, or null in a Nullable<bool> variable.

➔ Constructor- a method of a class, with the same name as the class, no specified return type, creates new objects of the class. There are two types of constructors:
- Default constructor: it has no parameter to pass.
- Parameterized constructor: it is invoked with parameters that are passed to the class during object creation.

➔ Code compilation in C# includes the following four steps:
1. Compile the source code in the managed code compatible with the C# compiler.
2. Combine the above newly created code into assemblies.
3. Load the CLR.
4. Execute the assembly by CLR to generate the output.

➔ Differences between Virtual Method and Abstract Method:
- Virtual method must have a default implementation, it can be overridden on the derived class, although it's not mandatory. Overridden using the keyword override
- Abstract method does not have implementation it resides in the abstract class, it is mandatory that the derived class implements the abstract method. The override keyword is not necessary although it can be used.

➔ The various methods of passing parameters in a method include:
- **Output parameters**: Lets the method return more than one value.
- **Value parameters**: The formal value copies and stores the value of the actual argument, which enables the manipulation of the formal parameter without affecting the value of the actual parameter.
- **Reference parameters**: The memory address of the actual parameter is stored in the formal argument, which means any change to the formal parameter would reflect on the actual argument.

➔ What is an array?
- An array is used to store multiple variables of the same type in a contiguous memory location.
- Arrays can be jagged, single or multi-dimensional.
  - Jagged arrays (array of arrays) is an array whose elements are arrays, it can be either single or multi-dimensional.
    ➔ Example:
    int[] jaggedArray = new int[4][];
  - Single dimension arrays are a linear array where the variables are stored in a single row.
    ➔ Example:
    Double numbers = new double[10];
    int[] score = new int[4] {25, 24, 23, 25};
  - Multi-dimensional arrays(rectangular arrays) have more than one dimension.
    ➔ Example:

```
int[,] numbers = new int[3, 2] {{1, 2}, {2, 3}, {3, 4}};
```

➔ Properties of an Array:
- Length: gets the total number of elements in the array
- IsFixedSize: tell whether the array is fixed in size or not
- IsReadOnly: tells whether the array is read-only or not

➔ Array Class: is the base class for all arrays, it provides many properties and methods and is present in the namespace system.

➔ Indexer - also known as an indexed property or smart arrays, an indexer is a class property allowing accessing a member variable of some class using features of an array. Used for treating an object as an array, an indexer allows using classes more intuitively. Defining an indexer enables creating classes that act like virtual arrays. Instances of such classes can be accessed using the [] array access operator.

➔ What is a string and its properties?
- A string is a collection of char objects
- Properties are:
  - Length: gets the number of objects in the current String
  - Char get the char object in the current string

➔ Some of the basic string operations are:
- Concatenate: Two strings can be concatenated either by using a System.String.Concat or by using + operator.
- Modify: Replace(a, b) is used to replace a string with another string.
- Trim() is used to trim the string at the end or at the beginning.
- Compare: System.StringComparison() is used to compare two strings, either a case-sensitive comparison or not case sensitive. Mainly takes two parameters, original string, and string to be compared with.
- Search: StartWith, EndsWith methods are used to search a particular string.

➔ What is parsing? How to parse a Date Time string?
- Parsing converts a string into another data type.

  Example:

  string text = "500";

  int num = int.Parse(text);

  500 is an integer. So, the Parse method converts the string 500 into its own base type, i.e. int.

  Follow the same method to convert a DateTime string.

  string dateTime = "Jan 1, 2018";

  DateTime parsedValue = DateTime.Parse(dateTime)

➔ Escape sequence is denoted by a backslash \ it indicates that the character that follows should be interpreted literally or it is a special character. An escape sequence is considered a single character. String escape sequences:
- \n – newline character
- \b – backspace
- \\ - backslash
- \' - single quote

- \" - double quote

➔ Regular expression is a template to match a set of input, the pattern can consist of operators, constructs, or character literals. Regex is used for string parsing and replacing the string character. Example:
- * matches the preceding character zero or more times. So, a*b regex is equivalent to b, ab, aab, aaab and so on.

```
Searching a string using Regex:
static void Main(string[] args)
{
string[] languages = { "C#", "Python", "Java" };
foreach(string s in languages)
{
if(System.Text.RegularExpressions.Regex.IsMatch(s,"Python"))
{
Console.WriteLine("Match found");
}
}
}
```

- The above example searches for "Python" against the set of inputs from the languages array. It uses Regex.IsMatch which returns true in case if the pattern is found in the input. The pattern can be any regular expression representing the input that we want to match.

➔ Exception handling is done using four keywords in C#:
- **try** – Contains a block of code for which an exception will be checked.
- **catch** – It is a program that catches an exception with the help of exception handler.
- **finally** – It is a block of code written to execute regardless whether an exception is caught or not. (Good place to keep closing connections to a database/releasing file handlers as it will execute at the end of the try-catch block)
- **throw** – Throws an exception when a problem occurs.

➔ Multiple catch blocks cannot be executed. Once the proper catch code is executed, the control is transferred to the finally block and the code that follows the finally block is executed.

➔ Differences between finally and finalize block:
- The finally block is called after the execution of try and catch block. It is used for exception handling. Regardless of whether an exception is caught or not, this block of code will be executed. Usually, this block will have a clean-up code.
- The finalize method is called just before garbage collection. It is used to perform clean up operations of Unmanaged code. It is automatically called when a given instance is not subsequently called.

➔ Difference between Continue and Break Statement
- Break statement breaks the loop. It makes the control of the program to exit the loop.
- Continue statement makes the control of the program to exit only the current iteration. It does not break the loop.

➔ Events are user actions that generate notifications to the application to which it must respond. The user actions can be mouse movements, key press, etc.
- Programmatically, a class that raises an event is called a publisher and a class which

responds/receives the event is called a subscriber. Events should have at least one subscriber else that event is never raised.

- Delegates are used to declare Events.
- Example:
  Public delegate void PrintNumbers();
  Event PrintNumbers myEvent;
- Publisher is a class that raises an event
- Subscribers respond/receive the event that it is interested in.

➔ Delegate is a variable that holds the reference to a method, it is a function pointer or function object. All delegates are derived from the System.Delegate namespace, delegates provide a form of encapsulation to the reference method which will get internally called when a delegate is called.

- Delegate allows a function to be passed as an argument to another function - it is a function object
- Declaring a delegate:
  - public delegate void AddNumbers(int n);
    // after the declaration of a delegate, the object must be created by the delegate using the new keyword
    AddNumbers an1 = new AddNumbers(number);
- Example:

```
public delegate int myDel(int number);
 public class Program
 {
 public int AddNumbers(int a)
 {
 int Sum = a + 10;
 return Sum;
 }
 public void Start()
 {
 myDel DelgateExample = AddNumbers;
 }
 }
```

➔ Different types of Delegates are:

- Single Delegate: A delegate that can call a single method.
- Multicast Delegate: A delegate that can call multiple methods. + and – operators are used to subscribe and unsubscribe respectively.
- Generic Delegate: It does not require an instance of the delegate to be defined. It is of three types, Action, Funcs and Predicate.
- Action– In the above example of delegates and events, we can replace the definition of delegate and event using Action keyword. The Action delegate defines a method that can be called on arguments but does not return a result
- *Public delegate void deathInfo();*
  Public event deathInfo deathDate;
  //Replacing with Action//
  Public event Action deathDate;
  - Action implicitly refers to a delegate.

- **Func**– A Func delegate defines a method that can be called on arguments and returns a result.
  - *Func <int, string, bool> myDel* is same as *delegate bool myDel(int a, string b);*
- **Predicate**– Defines a method that can be called on arguments and always returns the bool
  - *Predicate<string> myDel* is same as *delegate bool myDel(string s);*

➔ What are C# I/O classes? What are the commonly used I/O classes?

- C# has System.IO namespace, consisting of classes that are used to perform various operations on files like creating, deleting, opening, closing, etc.
- Whenever you open a file for reading or writing it becomes a stream which is a sequence of bytes traveling from source to destination. The two commonly used streams are input and output.
  - The stream is an abstract class that is the parent class for the file handling process.
  - The file is a static class with many static methods to handle file operation.
- Some commonly used I/O classes are:
  - FileStream – this stream reads from and writes to any location within a file
  - DirectoryInfo - perform operations on directories
  - FileInfo - perform operations on files
  - BinaryReader -read primitive data types from a binary stream
  - BinaryWriter - write primitive data types in binary format
  - File – Helps in manipulating a file.
  - StreamWriter – Used for writing characters to a stream.
  - StreamReader – Used for reading characters from a stream.
  - StringWriter – Used for writing a string buffer.
  - StringReader – Used for reading a string buffer.
  - Path – Used for performing operations related to the path information.
- What is StreamReader/StreamWriter class?
  - StreamReader and StreamWriter are classes of namespace System.IO. They are used when we want to read or write charact90, Reader-based data, respectively.
    - Some of the members of StreamReader are: Close(), Read(), Readline().
    - Members of StreamWriter are: Close(), Write(), Writeline().
    - Example:

```
Class Program1
{
using(StreamReader sr = new StreamReader("C:\ReadMe.txt")
{
//----------------code to read-------------------//
}
using(StreamWriter sw = new StreamWriter("C:\ReadMe.txt"))
{
//-------------code to write-------------------//
}
}
```

➔ Destructor is used to clean up the memory and free the resources. But in C# this is done by the garbage collector on its own. System.GC.Collect() is called internally for cleaning up. But sometimes it may be necessary to implement destructors manually.

➔ What are Boxing and Unboxing?

Converting a value type to reference type is called Boxing.
For Example:

```
int Value1 -= 10;
//————————Boxing————————————//
object boxedValue = Value1;
```

Explicit conversion of same reference type (created by boxing) back to value type is called Unboxing.
For Example:

```
//————————UnBoxing————————————//
int UnBoxing = int (boxedValue);
```

➔ Synchronization is a way to create a thread-safe code where only one thread can access the resource at any given time. The synchronous call waits for the method to complete before continuing with the program flow. Synchronous programming badly affects the UI operations when the user tries to perform time-consuming operations since only one thread will be used.

➔ In Asynchronous operations, the method call will immediately return so that the program can perform other operations while the called method completes its work in certain situations.

➔ Asynchronous programming means that the process runs independently of main or other processes. Example of Asynchronous

```
Public async Task<int> CalculateCount()
{
// write code to calculate Count of characters in a file
await Task.Delay(1000);
return 1;
}
public async Task myMethod()
{
Task<int> count = CalculateCount();
int result = await count;
}
```

- Async keyword is used for the method declaration.
- The count is of a task of type int which calls the method CalculateCount().
- Calculatecount() starts execution and calculates something.
- Independent work is done on my thread and then await count statement is reached.
- If the Calculatecount is not finished, myMethod will return to its calling method, thus the main thread doesn't get blocked.
- If the Calculatecount is already finished, then we have the result available when the control reaches await count. So the next step will continue in the same thread. However, it is not the situation in the above case where the Delay of 1 second is involved.

➔ A Thread is a set of instructions that can be executed, which will enable our program to perform concurrent processing which helps us do more than one operation at a time. By default, C# has only one thread, but other threads can be created to execute the code in parallel with the original thread.

- Threads are lightweight programs that save the CPU consumption and increase the efficiency of the application.

- Threads have a life cycle. It starts whenever a thread class is created and is terminated after the execution. *System.Threading* is the namespace which needs to be included to create threads and use its members.
- Threads are created by extending the Thread Class. *Start()* method is used to begin thread execution.

```
//CallThread is the target method//
ThreadStart methodThread = new ThreadStart(CallThread);
Thread childThread = new Thread(methodThread);
childThread.Start();
```

- C# can execute more than one task at a time. This is done by handling different processes by different threads. This is called MultiThreading.
- There are several thread methods that are used to handle multi-threaded operations**:**
  - Start
  - Sleep
  - Abort
  - Suspend
  - Resume
  - Join.

➔ Example Properties of thread class are:

- IsAlive – contains value True when a thread is Active.
- Name – Can return the name of the thread. Also, can set a name for the thread.
- Priority – returns the prioritized value of the task set by the operating system.
- IsBackground – gets or sets a value which indicates whether a thread should be a background process or foreground.
- ThreadState– describes the thread state.

➔ Different states of a thread are:

- Aborted – Thread is dead but not changed to state stopped
- Running – Thread starts execution.
- Stopped – Thread has stopped.
- Suspended – Thread has been suspended.
- Unstarted – Thread is created, but has not started to execute.
- WaitSleepJoin – Thread calls sleep, calls wait on another object and calls join on another thread.

➔ Race condition – when two threads access the same resource and try to change it at the same time. It is almost impossible to predict which thread succeeds in accessing the resource first. When two threads try to write a value to the same resource, the last value written is saved.

➔ A Deadlock is a situation where a process is not able to complete its execution because two or more processes are waiting for each other to finish. This usually occurs in multi-threading.

- Here a shared resource is being held by a process and another process is waiting for the first process to release it and the thread holding the locked item is waiting for another process to complete.

```
private static object objA = new object();
private static object objB = new object();
private static void PerformTaskA();
{
    lock(objB)
    {
```

```
            Thread.Sleep(1000);
            lock(objA)
            {
            }
        }
    }
    private static void PerformTaskB()
    {
        lock objA()
        {
            lock objB()
            {
            }
        }
    }
public static void Main()
{
    Thread thread1 = new Thread(PerformTaskA);
    Thread thread2 = new Thread(PerformTaskB);
    thread1.Start(); thread2.Start();
}
```

- Perform tasks accesses objB and waits for 1 second, meanwhile PerformTaskB tries to access objA. After 1 second, PerformTaskA tries to access objA which is locked by PerformTaskB, PerformTaskB tries to access objB which is locked by PerformTaskA.
- This creates a deadlock

➔ Circular reference is situation in which two or more resources are interdependent on each other causes the lock condition and make the resources unusable.

➔ Thread pool is a collection of threads. These threads can be used to perform tasks without disturbing the primary thread. Once the thread completes the task, the thread returns to the pool.

- System.Threading.ThreadPool namespace has classes that manage the threads in the pool and its operations.
  System.Threading.ThreadPool.QueueUserWorkItem(new
  System.Threading.WaitCallback(SomeTask));
- The above line queues a task. SomeTask methods should have a parameter of type Object.

➔ Generics or Generic class is used to create classes or objects which do not have any specific data type, the data type is assigned during runtime or when it is used in the program. Generics allow you to write a class or method that can work with any data type.

- Example:

```
class DataStore<T>
{
    public T Data {get; set;}
 }
 DataStore<string> store = new DataStore<string>();
```

➔ Get and Set are called Accessors: they are used by Properties. The property provides a mechanism to read and write the value of a private field. For accessing that private field,

accessors are used.

- Get Property is used to return the value of a property
- Set Property accessor is used to set the value.

➔ Serialization is a process of converting code to its binary format. Once it is converted to bytes, it can be easily stored and written to a disk or any such storage devices. Serializations are mainly useful when we do not want to lose the original form of the code and it can be retrieved anytime in the future.

- Any class which is marked with the attribute [Serializable] will be converted to its binary form.
- To Serialize an object we need the object to be serialized, a stream that can contain the serialized object and namespace System.Runtime.Serialization can contain classes for serialization.
- Binary Serialization – Faster and demands less space; it converts any code into its binary form. Serialize and restore public and non-public properties.
- SOAP – produces a complete SOAP compliant envelope that is usable by any system capable of understanding SOAP. The classes about this type of serialization reside in System.Runtime.Serialization.
- XML Serialization – Serializes all the public properties to the XML document. In addition to being easy to read, the XML document manipulated in several formats. The classes in this type of serialization reside in System.sml.Serialization.
- The reverse process of getting the C# code back from the binary form is called Deserialization.

➔ Difference between String and StringBuilder:

- String is an immutable object that holds string value, performance is slow, creates new instance to override or change the previous value. String belongs to the System namespace
- StringBuilder is mutable object, performance is fast, uses the same instance of StringBuilder to perform operation such as insert value in an existing string, belong to the System.Text.Stringbuilder namespace

➔ An enum is a value type with a set of related named constants often referred to as an enumerator list. The enum keyword is used to declare an enumeration. It is a primitive data type, which is user defined. An enum is used to create numeric constants in .NET framework. All the members of enum are of enum type. There must be a numeric value for each enum type.

- Enums are strongly typed constant, i.e. an enum of one type may not be implicitly assigned to an enum of another type even though the underlying value of their members are the same.
- Enum values are fixed. Enum can be displayed as a string and processed as an integer.
- The default type is int, and the approved types are byte, sbyte, short, ushort, uint, long, and ulong.
- Enums are value types and are created on the stack and not on the heap
- Every enum type automatically derives from System.Enum, thus we can use System.Enum methods on enums.

➔ Reflection - the ability of code to access the metadata of the assembly during runtime. A program reflects upon itself and uses the metadata to:

- Inform the user, or
- Modify the behavior
- The system contains all classes and methods that manage the information of all the loaded

types and methods. Reflection namespace. Implementation of reflection is in 2 steps:

- Get the type of the object, then
- Use the type to identify members, such as properties and methods

➔ XSD file - XSD denotes XML Schema Definition. The XML file can have any attributes, elements, and tags if there is no XSD file associated with it. The XSD file gives a structure for the XML file, meaning that it determines what, and also the order of, the elements and properties that should be there in the XML file. Note: - During serialization of C# code, the classes are converted to XSD compliant format by the Xsd.exe tool.

➔ Differences between ref and out keywords:

- In any C# function, there can be three types of parameters, namely in, out and ref. Although both out and ref are treated differently at the run time, they receive the same treatment during compile time.
- It is not possible to pass properties as an out or ref parameter.
- An argument passed as ref must be initialized before passing to the method whereas out parameter needs not to be initialized before passing to a method.

➔ A Singleton Design Pattern ensures that a class has one and only one instance and provides a global point of access to the same. There are numerous ways of implementing the Singleton Design Patterns in C#.

- Following are the typical characteristics of a Singleton Pattern:
  - A public static means of getting the reference to the single instance created
  - A single constructor, private and parameter-less
  - A static variable holding a reference to the single instance created
  - The class is sealed

➔ Extension Method – adds functionality to the extended class, add new methods to the existing ones, the methods that are added are static. At times, when you want to add methods to an existing class but don't perceive the right to modify that class or don't hold the rights, you can create a new static class containing the new methods. Once the extended methods are declared, bind this class with the existing one and see the methods will be added to the existing one.

➔ Difference between "is" and "as" operators in c#

- "is" operator is used to check the compatibility of an object with a given type, and it returns the result as Boolean.
- "as" operator is used for casting of an object to a type or a class.

➔ Difference between Equality Operator (==) and Equals() method:

- Equality operator (==) is a reference type which means that if equality operator is used, it will return true only if both the references point to the same object.
- Equality operator: Compares by reference
- Equals() method: Equals method is used to compare the values carried by the objects. int x=10, int y=10. If x==y is compared then, the values carried by x and y are compared which is equal and therefore they return true.
- Equals(): Compares by value

➔ Keywords in C#:

- Modifier Keywords:
  - **abstract** - modifier indicates that the thing being modified has a missing or incomplete implementation. The abstract modifier can be used with classes, methods, properties, indexers, and events
  - **async** - modifier to specify that a method, lambda expression, or anonymous method is asynchronous.
  - **const** -keyword to declare a constant field or a constant local. Constant fields and locals aren't variables and may not be modified. Constants can be numbers, Boolean values, strings, or a null reference.
  - **event** -keyword is used to declare an event in a publisher class
  - **extern** - modifier is used to declare a method that is implemented externally
  - **new** -operator creates a new instance of a type
  - **override** - modifier is required to extend or modify the abstract or virtual implementation of an inherited method, property, indexer, or event.
  - **partial** - Partial type definitions allow for the definition of a class, struct, interface, or record to be split into multiple files. A partial method has its signature defined in one part of a partial type, and its implementation defined in another part of the type
  - **readonly** -indicates that assignment to the field can only occur as part of the declaration or in a constructor in the same class. A readonly field can be assigned and reassigned multiple times within the field declaration and constructor.
  - **sealed** - modifier prevents other classes from inheriting from it
  - **static** -modifier to declare a static member, which belongs to the type itself rather than to a specific object. The static modifier can be used to declare static class
  - **unsafe** -keyword denotes an unsafe context, which is required for any operation involving pointers
  - **virtual** -keyword is used to modify a method, property, indexer, or event declaration and allow for it to be overridden in a derived class
  - **volatile**-keyword indicates that a field might be modified by multiple threads that are executing at the same time
- Access Modifier Keywords:
  - **public** – there are no restrictions on accessing public members
  - **private** – access is limited to within the class definition. This is the default access modifier type if none is formally specified
  - **protected** – access is limited to within the class definition and any class that inherits from the class
  - **internal** – access is limited exclusively to classes defined within the current project assembly
- Statement Keywords:
  - **if & else** – commonly used together although if can be used alone
  - **switch**-statement selects a statement list to execute based on a pattern match with a match expression,
  - **case** – used in switch statements, remember to use default case and break at the end
  - **do** -statement executes a statement or a block of statements while a specified Boolean expression evaluates to true.
  - **for** - statement executes a statement or a block of statements while a specified Boolean expression evaluates to true
  - **foreach** - statement executes a statement or a block of statements for each element in an instance of the type that implements the System.Collections.IEnumerable or System.Collections.Generic.Ienumerable<T> interface

- **in** - keyword is used in the following contexts:
  - generic type parameters in generic interfaces and delegates.
  - As a parameter modifier, which lets you pass an argument to a method by reference rather than by value.
  - Foreach statements.
  - from clauses in LINQ query expressions.
  - join clauses in LINQ query expressions.
- **while** -statement executes a statement or a block of statements while a specified Boolean expression evaluates to true. Because that expression is evaluated before each execution of the loop, a while loop executes zero or more times.
- **break** - statement terminates the closest enclosing iteration statement (that is, for, foreach, while, or do loop) or switch statement. The break statement transfers control to the statement that follows the terminated statement, if any.
- **continue** - statement starts a new iteration of the closest enclosing iteration statement (that is, for, foreach, while, or do loop
- **default** - keyword in the following contexts:
  - To specify the default case in the switch statement.
  - As the default operator or literal to produce the default value of a type.
  - As the default type constraint on a generic method override or explicit interface implementation.
- **goto** -statement transfers control to a statement that is marked by a label,
- **return** - statement terminates execution of the function in which it appears and returns control and the function's result, if any, to the caller
- **yield** - contextual keyword in a statement, you indicate that the method, operator, or get accessor in which it appears is an iterator
- **throw** - Signals the occurrence of an exception during program execution
- **try** - block contains the guarded code that may cause the exception. The block is executed until an exception is thrown or it is completed successfully
- **catch** - statement handles the exception
- **finally** - block, you can clean up any resources that are allocated in a try block, and you can run code even if an exception occurs in the try block. Typically, the statements of a finally block run when control leaves a try statement. The transfer of control can occur as a result of normal execution, of execution of a break, continue, goto, or return statement, or of propagation of an exception out of the try statement
- **checked** - keyword is used to explicitly enable overflow checking for integral-type arithmetic operations and conversions
- **unchecked** - keyword is used to suppress overflow-checking for integral-type arithmetic operations and conversions.
- **fixed** -statement sets a pointer to a managed variable and "pins" that variable during the execution of the statement.
- **lock**- statement acquires the mutual-exclusion lock for a given object, executes a statement block, and then releases the lock. While a lock is held, the thread that holds the lock can again acquire and release the lock. Any other thread is blocked from acquiring the lock and waits until the lock is released

◆ Method Parameter Keyword:

- **params** - keyword, you can specify a method parameter that takes a variable number of arguments. The parameter type must be a single-dimensional array
- **ref** - keyword indicates that a value is passed by reference. It is used in four different contexts:

    - In a method signature and in a method call, to pass an argument to a method by reference
    - In a method signature, to return a value to the caller by reference.
    - In a member body, to indicate that a reference return value is stored locally as a reference that the caller intends to modify. Or to indicate that a local variable accesses another value by reference.
    - In a struct declaration, to declare a ref struct or a readonly ref struct.
  - **out** - keyword in two contexts:
    - As a parameter modifier, which lets you pass an argument to a method by reference rather than by value.
    - In generic type parameter declarations for interfaces and delegates, which specifies that a type parameter is covariant.
- Access Keywords:
  - **base** - keyword is used to access members of the base class from within a derived class
  - **this** - keyword refers to the current instance of the class and is also used as a modifier of the first parameter of an extension method.
- Namespace Keywords:
  - **using** - keyword has two major uses:
    - The using statement defines a scope at the end of which an object will be disposed.
    - The using directive creates an alias for a namespace or imports types defined in other namespaces
  - **operator** -keyword to declare an operator. An operator declaration must satisfy the following rules:
    - It includes both a public and a static modifier.
    - A unary operator has one input parameter. A binary operator has two input parameters. In each case, at least one parameter must have type T or T? where T is the type that contains the operator declaration
  - **extern** alias -declaration introduces an additional root-level namespace that parallels (but does not lie within) the global namespace
- Literal Keywords:
  - **null** -keyword is a literal that represents a null reference, one that does not refer to any object
  - **false** - The default value of the bool type
  - **true** – bool value type
  - **void** - as the return type of a method (or a local function) to specify that the method doesn't return a value
- Operator Keywords:
  - **as** -operator explicitly converts the result of an expression to a given reference or nullable value type
  - **await** - operator suspends evaluation of the enclosing async method until the asynchronous operation represented by its operand completes. When the asynchronous operation completes, the await operator returns the result of the operation, if any
  - **is** - operator checks if the result of an expression is compatible with a given type
  - **sizeof** - operator returns the number of bytes occupied by a variable of a given type
  - **typeof** - operator obtains the System.Type instance for a type. The argument to the typeof operator must be the name of a type or a type parameter
  - **stackalloc** - expression allocates a block of memory on the stack. A stack allocated memory block created during the method execution is automatically discarded when that method returns. You cannot explicitly free the memory allocated with stackalloc. A

stack allocated memory block is not subject to garbage collection and doesn't have to be pinned with a fixed statement

- Contextual Keywords:
  - **add** -contextual keyword is used to define a custom event accessor that is invoked when client code subscribes to your event.
  - var -
  - dynamic -
  - **global** - namespace is the namespace that contains namespaces and types that are not declared inside a named namespace.
  - **set** - keyword defines an *accessor* method in a property or indexer that assigns a value to the property or the indexer element
  - **value** - references the value that client code is attempting to assign to the property or indexer
- Type Keywords:
  - **bool** - type keyword is an alias for the .NET System.Boolean structure type that represents a Boolean value, which can be either true or false
  - **char** - type keyword is an alias for the .NET System.Char structure type that represents a Unicode UTF-16 characte
  - class -
  - decimal -
  - double -
  - **enum** - is a value type defined by a set of named constants of the underlying integral numeric type.
  - **float** - represent real numbers. All floating-point numeric types are value types
  - **string** - type represents a sequence of zero or more Unicode characters
  - **struct** -is a value type that can encapsulate data and related functionality
- Query Keywords:
  - **from** - A query expression must begin with a from clause The from clause specifies the following:
    - The data source on which the query or sub-query will be run.
    - A local range variable that represents each element in the source sequence
  - **where** - clause is used in a query expression to specify which elements from the data source will be returned in the query expression. It applies a Boolean condition (*predicate*) to each source element (referenced by the range variable) and returns those for which the specified condition is true
  - **where** - clause in a generic definition specifies constraints on the types that are used as arguments for type parameters in a generic type, method, delegate, or local function.
  - **select** -clause specifies the type of values that will be produced when the query is executed.
  - **group** - clause returns a sequence of Igrouping<Tkey,Telement> objects that contain zero or more items that match the key value for the group.
  - **into** - contextual keyword can be used to create a temporary identifier to store the results of a group, join or select clause into a new identifier. This identifier can itself be a generator for additional query commands
  - **orderby** -clause causes the returned sequence or sub-sequence (group) to be sorted in either ascending or descending order.
  - **join** - clause is useful for associating elements from different source sequences that have no direct relationship in the object model.
  - **let** - creates new range variable & initializes it with the result of the expression you

supply

- **on** -contextual keyword is used in the join clause of a query expression to specify the join condition
- **equals**- contextual keyword is used in a join clause in a query expression to compare the elements of two sequences.
- **by**- contextual keyword is used in the group clause in a query expression to specify how the returned items should be grouped.
- **ascending**- contextual keyword is used in the orderby clause in query expressions to specify that the sort order is from smallest to largest
- **descending**- contextual keyword is used in the orderby clause in query expressions to specify that the sort order is from largest to smallest.

➔ Mathematical keywords

C# type/keyword	Range	Size	.NET type
sbyte	-128 to 127	Signed 8-bit integer	System.SByte
byte	0 to 255	Unsigned 8-bit integer	System.Byte
short	-32,768 to 32,767	Signed 16-bit integer	System.Int16
ushort	0 to 65,535	Unsigned 16-bit integer	System.UInt16
int	-2,147,483,648 to 2,147,483,647	Signed 32-bit integer	System.Int32
uint	0 to 4,294,967,295	Unsigned 32-bit integer	System.UInt32
long	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	Signed 64-bit integer	System.Int64
ulong	0 to 18,446,744,073,709,551,615	Unsigned 64-bit integer	System.UInt64
nint	Depends on platform (computed at runtime)	Signed 32-bit or 64-bit integer	System.IntPtr
nuint	Depends on platform (computed at runtime)	Unsigned 32-bit or 64-bit integer	System.UIntPtr