FortuneCloud
We Understand Technology

C# language

Introduction to C#

- Basic Structure of C#
- What is Namespace
- Purpose of Main Method

What is DLL Hell?

Before some time, if we install an application then dll of that application get stored in the registry, then if we install other application that has same name .dll that means previously installed .dll get overwrite by the same name new .dll. Ok for newly installed application but previously installed application cant get execute further. This is big problem in context of version of same application. This is Dll-Hell problem.

How Solve by .Net Framework:

Net Framework provides operating systems with a Global Assembly Cache. This Cache is a repository for all the .Net components that are shared globally on a particular machine. When a .Net component is installed onto the machine, the Global Assembly Cache looks at its version, its public key, and its language information and creates a strong name for the component. The component is then registered in the repository and indexed by its strong name, so there is no confusion between different versions of the same component, or DLL

OOPs Concept :

Class:

A class is a blueprint of an object that contains variables for storing data and functions to perform operations on the data.

A class is a collection of the object.

A class will not occupy any memory space and hence it is only a **logical representation of data.**

It is a user defined data type.
It is a reference type

Object:

An object is any entity that are having state (i.e. variable) & behavior(i.e. method).

An object is an instance of the class.

A class will not occupy any memory space. Hence to work with the data represented by the class you must create a variable for the class, that is called an object.

Abstraction:

Abstraction is the process of hiding the implementation details & showing the necessary/essential features.

Abstraction lets you focus on what the object does instead of how it does it.

Abstraction provides you a generalized view of your classes or objects by providing relevant information.

Abstraction means putting all the variables and methods in a class that are necessary.
Ex. Abstract class & abstract method.

Examples. Mobile Phones – does call & msg to another but hides the how it does?

Example. Application fill by doctor & student. Use of abstract class

Encapsulation:

Encapsulation is the process of binding the data members & data function into a single unit.

Encapsulation is like enclosing in a capsule. That is enclosing the related operations and data related to an object into that object.

Encapsulation prevents clients from seeing its inside view, where the behaviour of the abstraction is implemented.

Encapsulation is a technique used to protect the information in an object from another object.

Examples: Mobile Phone & its Manufactures
Ex. TV Operation using remote.

Encapsulation is implemented by using access specifiers. An access specifier defines the scope and visibility of a class member. Available access specifiers are public, private, protected, internal etc.

Inheritance:

It is the process of acquiring the data members & functions of one class by another class.

It is totally for object reusability.

One class can include the feature of another class by using the concept of inheritance

The main advantage of extending classes is that it provides a convenient way to reuse existing fully tested code in different context thereby saving lots of time with existing coding and its model style

Example: Parent child relation
Example: Bike manufactuers – new version of bike – used old set up with some modification.

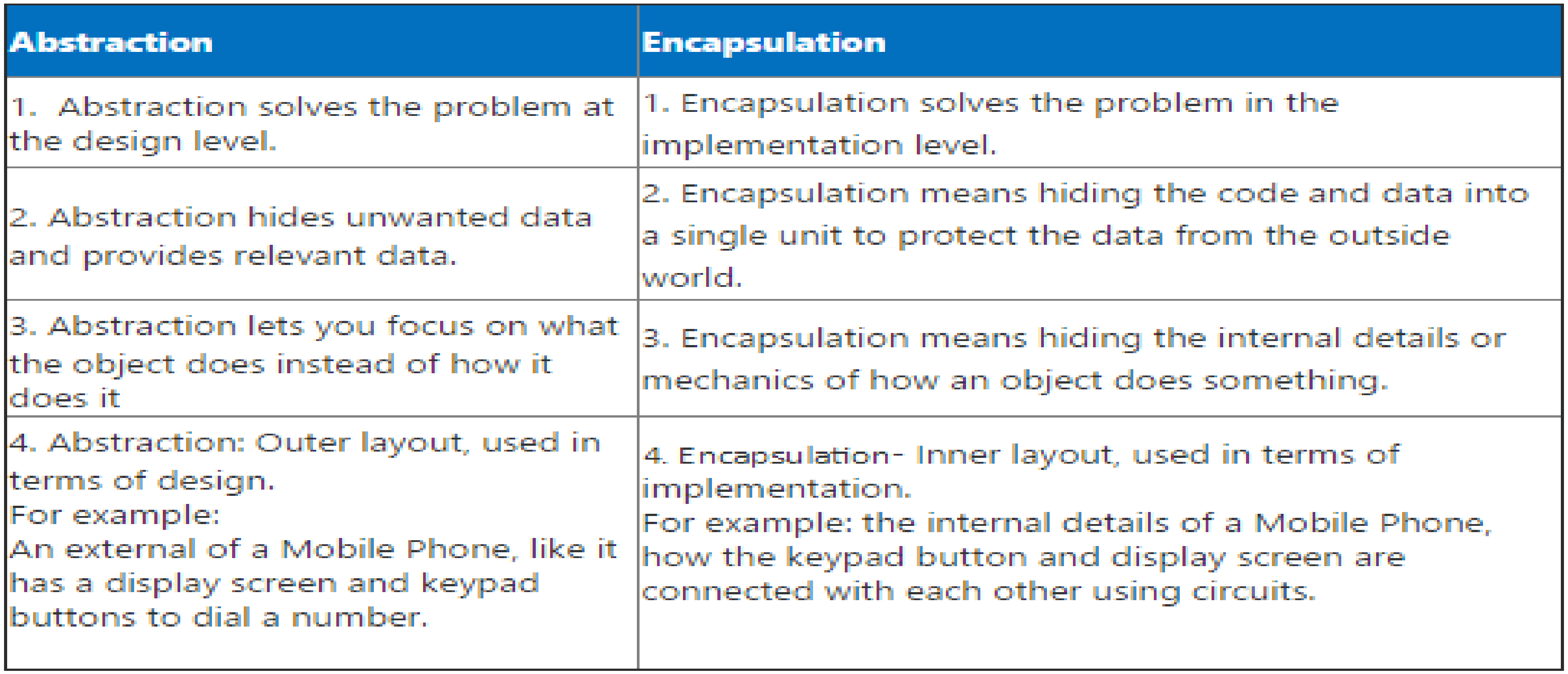
Polymorphism:

Poly means many i.e. one thing can behave /perform differently in different situation.

Example:

1. A man role can change from home, office.
2. Mobile phone , as a calling , as a camera etc

Difference between Abstraction & Encapsulation:



Abstraction	Encapsulation
1. Abstraction solves the problem at the design level.	1. Encapsulation solves the problem in the implementation level.
2. Abstraction hides unwanted data and provides relevant data.	2. Encapsulation means hiding the code and data into a single unit to protect the data from the outside world.
3. Abstraction lets you focus on what the object does instead of how it does it	3. Encapsulation means hiding the internal details or mechanics of how an object does something.
4. Abstraction: Outer layout, used in terms of design. For example: An external of a Mobile Phone, like it has a display screen and keypad buttons to dial a number.	4. Encapsulation- Inner layout, used in terms of implementation. For example: the internal details of a Mobile Phone, how the keypad button and display screen are connected with each other using circuits.

"Encapsulation is accomplished using classes. Keeping data and methods that access that data into a single unit."
"Abstraction is accomplished using an Interface. Just giving the abstract information about what it can do without specifying the details.

Difference between DLL & EXE:

1. EXE is an extension used for executable files while DLL is the extension for a dynamic link library.
2. An EXE file can be run independently while a DLL is used by other applications.
3. An EXE file defines an entry point while a DLL does not.
4. A DLL file can be reused by other applications while an EXE cannot.
5. A DLL would share the same process and memory space of the calling application while an EXE creates its separate process and memory space.

History

• Design & developed by Anders Hejlsberg during the development of .Net Framework.

•Base used is C & C++ , initialy named as COOL C like Obejct Oriented Language but later comes with C# due to trademark.

•C# Name is inspired from musical notes & sharp notes indicates that this note is higher than other notes.

Release in 2001.

Sample Program

```
// Namespace declaration
Using System;
Class Program;
 {
     static void Main(strings[]args)
       {
         Console.WriteLine("Hello
            World");
        }
}
```

Using System Declaration

The namespace declaration indicates that you are using System Namespace.

A Namespace is used to organise your code and is a collection of classes, interfaces, structs,enums and delegates.

Main method is an Entry Point to your application

Reading & Writing to Console

- Reading from console
- Writing to console
- 2 Ways to Write to Console

a) Concatenation
b) Using Placeholder Syntax- Most preferred.

Variables & Data Types in C#:

Variable allows you to store value and reuse it later in the program.
Variable is an entity whose value can keep changing.
In c# variable is a location in computers memory that is identified by unique name and used to store value.
Different type of data such as character, integer or a string can be stored in variable.

Based on the types of data to be store in variables, variables can be assign different data types.

<u>Syntax:</u>

<datatype><Variable Name>;

Where datatype = valid c# datatypes
Variable name = Is a valid variable name

<u>Variable Initialisation:</u>

<variable name> = value;
Where = is the assignment operator used to assign value.

Value = Is the data that can be store

Examples : int Empno, string Empname;

```
        Empno = 12;
        Empname = ‘Pritam’
```

Rules to declare variables:

1. Letter , digit, underscore – ok
2. Use digit as a first character – not ok
3. C# is a case sensitive, so variables with uppercase and lowercase are different.
4. Use of c# keywords – not recommended but if u want to use prefix it with @ symbols.

Microsoft recommended camelCase notation for variable naming

In c# ,you can declare multiple variables in the same way as single variables.

DataTypes: To identify type of data that can be store in variable , C# provides different types of data types.

Data Types Divided into 2 categories.

1. Value Types
2. Reference Types

Variables of Value types store the actual values. These values store over the Stack memory.

These values can be either built in datatypes or user defined data types.

Default Value of Value types is 0, i.e it always has value

Stack storage leads to faster access to variables than reference types.

i.e. No heap allocation, less gc pressure. More efficient use of memory.

Variables of Reference Types store the memory address of other variables in Heap Memory.

Default value of reference type is null.i.e. it may contain null.

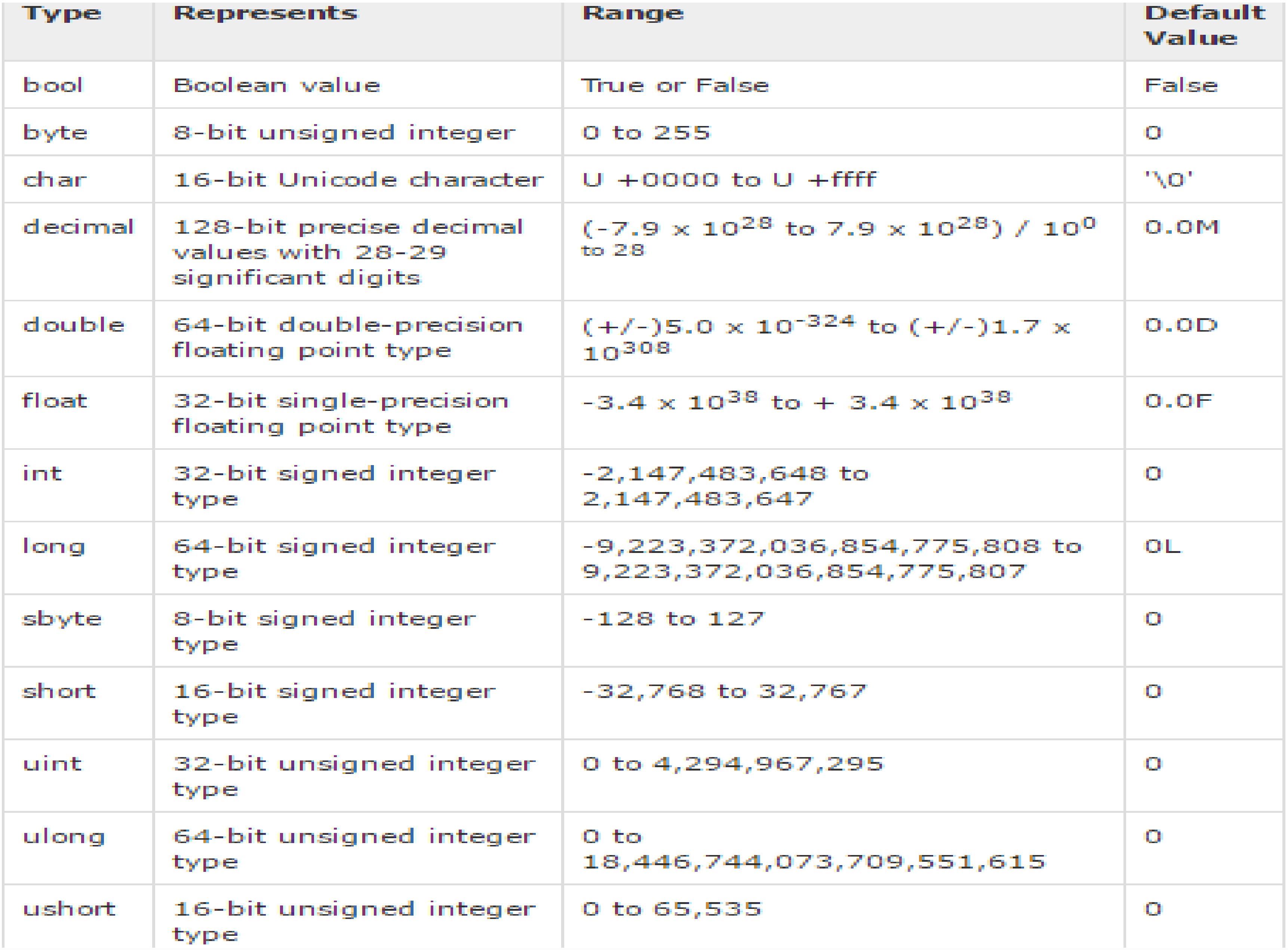
Predefined Data Types:

It is referred as basic data types in c#. These data types have predefined size and ranges.

Size of the data types helps the compiler to allocate the memory.
And
Range of the data types helps compiler to ensure that the value store is within the range of the datatypes.

User Defined :

1. Struct,
2. Enum



Type	Represents	Range	Default Value
bool	Boolean value	True or False	False
byte	8-bit unsigned integer	0 to 255	0
char	16-bit Unicode character	U +0000 to U +ffff	'\0'
decimal	128-bit precise decimal values with 28-29 significant digits	(-7.9 x 10^{28} to 7.9 x 10^{28}) / 10^{0} to 28	0.0M
double	64-bit double-precision floating point type	(+/-)5.0 x 10^{-324} to (+/-)1.7 x 10^{308}	0.0D
float	32-bit single-precision floating point type	-3.4 x 10^{38} to + 3.4 x 10^{38}	0.0F
int	32-bit signed integer type	-2,147,483,648 to 2,147,483,647	0
long	64-bit signed integer type	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0L
sbyte	8-bit signed integer type	-128 to 127	0
short	16-bit signed integer type	-32,768 to 32,767	0
uint	32-bit unsigned integer type	0 to 4,294,967,295	0
ulong	64-bit unsigned integer type	0 to 18,446,744,073,709,551,615	0
ushort	16-bit unsigned integer type	0 to 65,535	0

References Data Types: It store the memory reference of other variables and other variables store the actual data.

Types of reference data types

1. Built In Reference Types
   ex. Object , string.

2.User Defined Reference Types
   ex. Class , interface , delegate ,array

Comments in C#: Comments are given by the programmer to increase the readability of program.
Comments are ignored by the compiler at the time of compilation.

Types

1. Single Line comment   //
2. Multiple Line comment   /* … */

Constant in C#:

Constant in C# are fixed values assigned to the identifier that are not modified throughout the execution of program.

Use: Constant are defined to when you want to preserve that value to reuse them later or to prevent any modification to the values.

In c# you can declare constant to any data types.

Constant are declare for value types rather than reference types.

Declaration:

const <DataType><identifier name> = <value>

Const is keyword ,

Example: Area of circle

Literals :

Literal is a static value assigned to variables and constant.

You can defined literals to any DataType in c3

Examples : string bookname = “C#”;

Here “C#” is a literal assigned to variable bookname of type string

Types :

1. Boolean Literal : It has two values true or false
   bool val =true;
2. Integer Literal: It assigned to int ,uint,long,ulong with suffixes
   L,UL,
   long val =55L;
3. Real Literal : It assigned to float , double(default) , decimal with suffixes
   F,D,M
   float val = 2.35F.
4. Character Literal : It assigned to char datatypes always enclose with single quote; char val=‘A’;

5.String Literal : There are two types of string literal

1. Standard string
2. Verbatim string

A regular string literal is standard string

A verbatim string literal is similar to standard string literal but it is prefixed by @ symbols

It is always enclosed with double quotes ";

string maildomain ="@gmail.com"

Example of verbatim string literal.

6. Null Literal : The null literal has only one value i.e. Null

string email = null;

Where Null specifies that email does not belong to any objects or reference.

Keywords:

Keywords are reserved words and separately compile by the compiler. They convey predefined meaning to compiler and cannot be created or modified.

You can not used the keyword as class name, variable name.

If you want to do that you should use prefix @ character.

Examples: int , double, namespace , using , etc.

Need of Escape Sequence Character :

Example of Payroll System.

Escape Sequence Character:

It is an special character that is prefixed by backslash \.
It is used to implement special non printing character such as new line, single space or backspace.
The non –printing character is used while displaying the formatted output increasing the user’s readability.

There are various sequence character in c# as follows

Operators in C#:

An operator is an symbol that tells the compiler to perform specific mathmathical or logical operation or manipulation.

C# has rich set of instruction.

1. Arithmetic Operator
2. Relational
3. Logical
4. Conditional
5. Increment and Decrement
6. Assignment Operator.

1. Arithmetic Operator:

It is binary operator because they work with 2 operands.

C# arithmetic Operators are

+	Addition
-	Sub
*	Multiplication
/	division
%	module

2. Relational Operator:

It is also a binary operator , works on 2 operator but here it compares two operands and return boolean value true of false.

C# relational Operators are :

==	equality
!=	inequality
>	Greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

3. Logical operators:

it is a binary operator that performs logical operation on two operands and return a boolean value.

C# supports two types of Logical operator

a. Boolean
b. Bitwise

Boolean operators:

&&	Boolean And
||	Boolean Or
^	Exclusive Or
!	Complement or negation

4. Assignment Operator:

It is used to assign the value of right side operand to the value of left side operand using = operator

Types

a. Simple assignment operator : It uses only = operator
b. Compound assignment operator : It uses = operator and arithmetic operator

5.Increment & Decrement Operator :

In C# ++ increment operator is used to increase the value by 1 while decrement operator is used to decrease the value by 1.

Pre-Increment/ Decrement : If operator is placed before the operand, expression is called as pre increment. Decrement.

Post-Increment/ Decrement: If operator is placed after the operand.

TypeCasting : Casting is converting data from one form to another form

Types of Casting:

1. Implicit Casting : It is done automatically by CLR but both the datatypes belong to same hierarchy . Also destination datatypes hold the a larger range of values than the source data types.

   It prevents the loss of data as destination datatypes hold the larger values.

2. Explicit Casting: It converts the data types of Higher precision to data types of lower precision.

   This typecasting might result in loss of data.

Syntax:

```
<Destination Dt><variable name> =
                                    (destination dt)<source datatypes>
```

Explicit TypeCasting Can be done in two ways using built in methods:

a. Using System.Convert class
b. Using ToString() method : this method belongs to object class.

Object Class is base class for all value types created in c#.

System is base namespace for all namespaces in C#.

Casting is on the basis of range not on size.

Explain the Range diagram

Boxing & Unboxing :

Boxing is the process of transforming from value type to reference type(object only). The runtime creates temparory box on heap for object.

Unboxing is the process of transforming from reference type to value type. Explicit Casting needed.

Syntax for Boxing:

Object<instance Object Class> = <Variable of value type>

Syntax for Unboxing:

<target Value type> <variable name> = (target value type) Object type.

Notes: We can not perform any arithemathic or other operation on object type , it just store the value temporarily.
Example: Courier parcel services

Notes:

1. Boxing and Unboxing used only in a situation where until runtime we don’t know the type of data we are going to deal with.
2. Excessive usuages of Object data types makes the languaged “loosely typed”. Also because of frequent casting requirement , performance degrades.

Decision Making Statement:

1.While loop
2.Do while loop
3.For loop
4.For each loop

1.While loop:

Syntax:
```
initialise loop counter;
  while(test loop counter using condition)
{
  statements;
}
```

2.Do while:

Syntax :
```
do
 {
    statements;
    }
     while(conditions);
```

Difference between while loop and do while loop:
1 . Nearly same as while but guaruntide to execute statements at once.
2. In do while condition comes after statements.
3. While loop also called as pre test loop ; and do while as post check loop.

3.For Loop: similar to while statement as condition check before the loop

```
for( initialisation statement ; test expression ; update statement)
{
   codes to be executed;
}
```

Nested For loop: It is important to understand.

4.For each :

Syntax :

```
     foreach (<datatype ><identifer>in<list>)
{
  one or more statements;
}
```

Difference between for loop & foreach loop?

Array:

An array is a collection of similar elements i.e. datatypes. i.e. all of int , all of float.

An array is a collection of related values placed in contiguious memory location and these values are reference using common array name.

Example : 100 student names.
Array: As array is reference type array, whose creation needs 2 steps.

Declaration:

```
type[] arrayname;
```

Where type is datatypes i.e. int, float etc.
Arrayname is idenfier given to array.

Initialisation:

```
type[]arrayname={val1,val2,…….val n};
```

Using new keyword:

type [] arrayname = new type[size-value]

```
Int[]nos=new int[10];
Initialisation  nos[0] = 10;
                nos[1] =20;
```

Array:

Types of array:

1. Single Dimensional array
2. Multi Dimensional array

a) Rectangular Array
b) Jagged Array.

Rectangular Array: 2 dimentional array is nothing but the list of array.

Array Declaration:

<datatype> [,] arrayname = new datatype [3 ,2] example

```
 int [ , ] numbers = new int [ 3 , 2]
```

Initialiazing 2 dimension array:

Multidimensional arrays may be initialized by specifying bracketed values for each row. The Following array is with 3 rows and each row has 4 columns.

```
  int [,] a = new int [3,4] {{0, 1, 2, 3},{4, 5, 6, 7},{8, 9, 10, 11}};
```

Jagged Array:

It is an array whose elements are also arrays. The element of jagged arrays can be of different dimensions and sizes.

It is also called as Array of array.

Jagged array declaration :

```
 int [][]arrayname = new int [3][];
```

Initialisation:

```
 Arrayname[0]= new int[5] or
 Arrayname [0] = new int[]{2,3,5,6,7}
 Arrayname[1]=new int[]{1,2,2}
 Arrayname[2] = new int[]{11,12}
```

OOPs Concept:

What is object oriented programming? And why it is better than procedure oriented programming.

Procedural Language – focus on ways to manipulate data
Object oriented - focus on data itself.

Object oriented programming has no of features as

1. Abstraction ex. Tv and manual
2. Encapsulation ex. ATM m/c
3. Inheritance ex. Human and child
4. Polymorphism ex. Oven.

Abstraction: It is a feature of abstracting only the required information from the object.
Encapsulation: It is a feature of data hiding or it can display only specific information to user and other information is hidden.
Inheritance: It is process of creating new class using or based on existing class attributes and methods. It helps to reuse the inherited attribute and methods.

Polymorphism:

It is ability to behave differently in different situition. It is mostly seen where you have multiple methods with same name and different parameters and different behavior.

Object:

Any entity that has identity, state and behavior.

Identity of object distinguishes it from other object of same type.

State of object refer to its characteristics or attributes.

Behavior of object comprises its action

The object stores its identity and state in a variable / attribute and exposes its behavior through method.

Examples: a car,a persons, etc.

Class:

several objects has a common state and behavior & thus can be grouped under a single class.

i.e. **Class is nothing but the collection of object.**

Example: class car contains object nissan , rand rover, bMW,Audi

Concept of class in real time belong to programming world in object oriented language. **A class is a template or blueprint which defines the state and behavior of all object belonging to that class.**

A Class is composed of field/attributes/variables and methods collectively called as data member of class.

Syntax to declare class :

```
      class <classname>
{
  // data members of class
}
```

Where classname specifies the name of class.

Rules/ conventions to define classes:

1. Should be noun
2. Cannot be of mixed case and first letter should be capitalise.
3. Should be simple , descriptive and meaningfull
4. Can not be a keyword.
5. Can not begin with digit may begin with @ or _ symbols.

Examples: valid class name: Account,_account,@account
Invalid class name: 2account,Acc,aCount, class.

Instantiating Object:

It is necessary to define the object of the class to access the variables and methods of that class.

Syntax:

```
<classname> <objectname> = new <classname>();
```

where on encountering the new keyword JIT allocates memory for the object and return the reference to the object.

Methods:

Methods are nothing but the function declare in a class and may be used to perform operation on class variables.
That can take one or more input and may or may not return value.

Methods can be invoked using the object.

Example : car as a class and break () method.

Convention or rules to specify the Names of Method:
1. Can not be c# keyword.
2. Can not contain spaces.
3. Can not begin with digit.
4. Can begin with letter ,@ or _.

Examples :
Syntax:

```
<access modifier><return type><method name>([List of parameters])
 {

  // Body of the method

 }
```

Where,
Access modifier: It specifies the scope of access of method. If no access modifier specified then it is treated as private as default
Return type: it is optional if method not returning any value then use void.
Name of Method :
List of parameters: It specifies the no of argument to be pass to method.

Invoking Method:

You can invoke method in a class by creating object of the class.

In c# mainly method is invoked or called from another method. Calling method and called method.

Syntax : Objectname . methodname (parameter list);

Static Methods:

A method is by default invoke or called without using the object of the class.

However it is possible to call method without using the object of the class.

This can be done declaring the method as static

Example as Main method is a static method hence it does not require object to invoke it.

Static method can refer to only static variables and other static method of the class.

However Static method can refer to non static method and variables using the instance of the class.

Syntax:

```
        static <return type><method name>()
    {
       body of method
     }
```

Static variables :

Static variable is a special variable that is access without using the object of the class.

When a variable is created as a static , it is automatically initialized before it access.

An instance of the class can not access the static variables.

Also called as class variable.

Only one copy of static variable is shared by all object of the class so change in such a value will reflect in all objects.

Access Modifiers:

A access modifier is a keyword of the c# language that is used to specify the access level of the members of the class.

It allows to specify which classes can access the members of other classes.

In C# 5 types

1.Public – from class to any other class – most permissive
2.Private – only within the class where they declare – least permissive
3.Internal – access is limited to current assembly only
within the assembly of the class.
4.Protected – any class that inherited from other class
5.Proteced Internal - Current assembly and derived classes
Within the class, subclass of that class
and assembly

Ref and out keyword can be used as a part of signature in overloaded methods.

this Keyword:

" this" keyword is used to refer to the current object of the class.

It is used to resolve the conflicts between the variables having the same names and to pass the current object as a parameter.

You can not used the this keyword with static variables and methods. Because this keyword refer to current object of the class & static means no object.

We can't access a field variable without this keyword in C#. this always refer to the members of same class.

Constructor:

A special method of the class that will be automatically invoked when an instance of the class is created is called a constructor.

The main use of constructors is to initialize private fields of the class while creating an instance for the class.

When you have not created a constructor in the class, the compiler will automatically create a default constructor in the class.

The default constructor initializes all numeric fields in the class to zero and all string and object fields to null.

Notes:

1. A class can have any number of constructors.
2. A constructor doesn't have any return type, not even void.
3. A static constructor can not be a parameterized constructor.
4. Within a class you can create only one static constructor.

Constructor has not a return type because implicit return type of constructor is class itself.

Types of Constructor:

1.Default Constructor
2.Parametrized Constructor

3.Copy Constructor
4.Static Constructor
5.Private Constructor

For constructor default access modifier is public. We can change it to private.

Overloading of Constructor:

The concept of declaring more than one constructor in a class is called as Constructor overloading.

Destructor:

It is a special method which has same name as class but Start with the character ~ before the classname.

Destructor immedialty deallocates the memory of object that are no longer used.They are invoke automatically.

You can define only one destructor in a class.

Notes:
1.Destructor can not be overloaded or inherited.
2. Destructor can not explicitly invoked
3. Destructor can not specifiers access modifier and can not take parameters.

Inheritance:

It is the process of creating new class from existing class. Or
It is the process of acquaring the methods and attributes of the base class.

The class from which the new class is created is called as Base class. The new class is called as derived class.

Example: parent & child , Vehicles

Syntax:

<derivedClassname>:<baseclassname>

Purpose of Inheritance:

1. **Reusability** :The purpose of inheritance is reused the common methods & attributes among classes without recreating them.

Example of class: Animal & cats.

2. **Generalisation:** Inheritance allows you to implement generalisation by creating base classes.

Examples : Vehicles and other specialised class.

3. **Specilisation:** Inheritance allows you to implement specilisation by creating derived class. Ex. Vehicles as derived classes truck, bus.

4.Extension : Inheritance allows you to extend the functionality of derived class by creating more methods & attributes than base class.

Implementing Inheritance:
Derive class can inherits the non-private attributes & methods of the base class.
Protected Access Modifier:
Variables & methods that are declared as protected are access only by that class and derived class.
Syntax :
Use protected keyword before variable or methods.

Base Keyword:
The base keyword allows you to access the variables & methods of the base class from the derived class.
Syntax:

New Keyword:
The new keyword can either acts as an operator or as an modifier.

Method Overloading:

Real Life Example : Hotel or resort having peter as a chef and also a maintaince incharge.

While declaring the method , the signature of method is written in {} next to method name.

No class is allowed to have two methods with same name but class can contain same method with different signature.

The concept of declaring more than one method with the same method name but different signatures is called as Method Overloading.

Signature of these methods are said to be same if Condition:

1. **No of parameters pass to the method**
2. **The parameters type**
3. **The order in which parameters type.**

return type is not a part of signature

A method cannot be overloaded on the basis of

1. Different return type
2. Different access modifier
3. Normal and optional parameters

Method Overriding:

It is the feature that allows the derived class to override or redefine the method of the base class.

Overriding of the method in the derived class can change the body of the method that was declare in the base class.

Thus the same method name with same signature declare in the base class can be reused in the derived class to define the a new behaviour.

This is how reusability ensures while inheriting the class.

The method implemented in the derived class from the base class known as overridden method.

While overloading consider the accesibility, scope of the method.
Ex. Private method in the base class can not be overriden as public in derived class.

Virtual & Override:

A method declared using virtual keyword is called as virtual method.

It is mandatory to use Override keyword in the derived class for method overriding.

Syntax for declaring virtual method

<accessmodifier>Virtual<return type><methodname><[parameters list]>

Syntax for declaring override method

<accessmodifier>overrride<returntype><methodname><parametrlis>

Conditions:

1. If derived class tries to override the non virtual method , compiler generates error.
2. If you fail to override the virtual method compiler shows the warning and code run successfully.
3. You can not used the keywords new , static , virtual with the override keyword.

Calling the base class method:

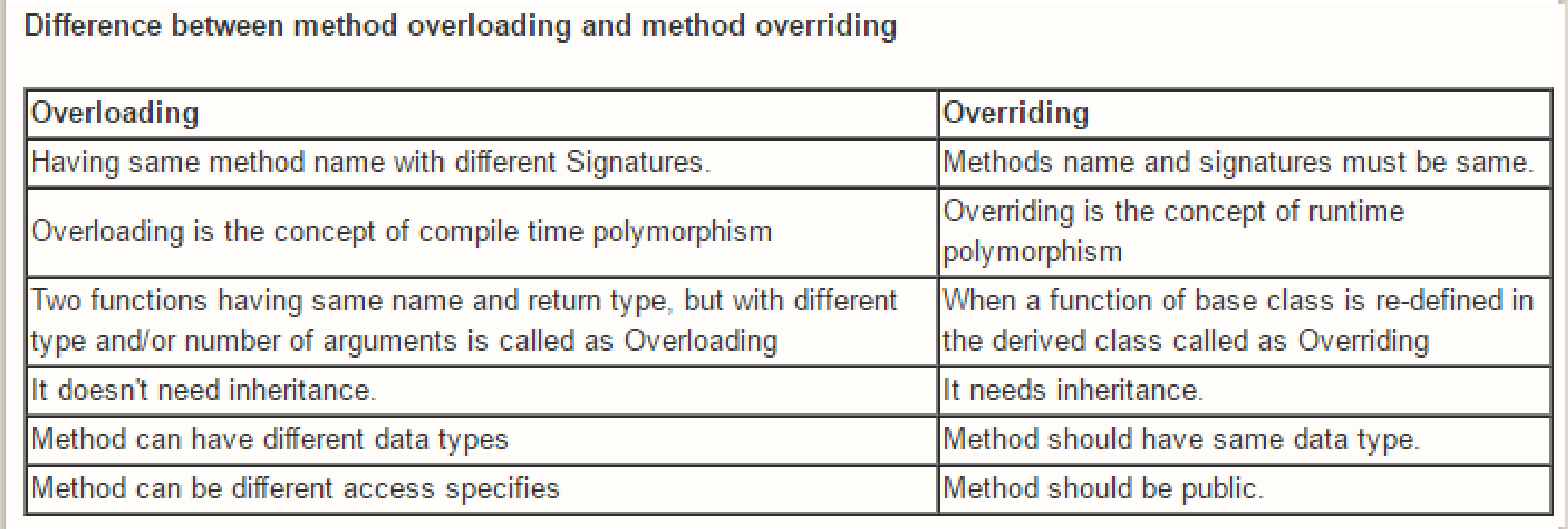
if I want to call base class method & overriden method then use the instance of the base class method to call & derived to derived one.

Example:

Difference between method Overloading & Overriding:

Overloading can be done in same class
Overriding can be done in parent and derived class
Overloading in used when we need same method in same class with different parameters
Overriding is used when we need to redefine a method that has already been defined in parent class (using the exact same signature
Overloading is resolved at compile time
Overriding is resolved at run time

When overriding, we change the method behaviour for a derived class.
Overloading simply creating more than one method with the same name but different parameters in same class.



Difference between method overloading and method overriding

Overloading	Overriding
Having same method name with different Signatures.	Methods name and signatures must be same.
Overloading is the concept of compile time polymorphism	Overriding is the concept of runtime polymorphism
Two functions having same name and return type, but with different type and/or number of arguments is called as Overloading	When a function of base class is re-defined in the derived class called as Overriding
It doesn't need inheritance.	It needs inheritance.
Method can have different data types	Method should have same data type.
Method can be different access specifies	Method should be public.

Sealed Class:

A sealed class is a class that prevents the inheritance. You can declare the sealed class by preceding the class keyword with the sealed keyword.

The seal keyword can not be base class as it prevent the class from inheriting.

Syntax to declare seal class:

```
sealed  class <classname>
{
 // body of the class.
 }
```

Notes:

1. Sealed class can not have a protected member if you tries to declare it, compiler generates warning only but can not access the protected member. As sealed class prevent the Inheritance.

Purpose of the Sealed class:

1. To ensure security
2. Protect from copyright problems.
3. Prevent Inheritance.

Guidelines when to apply sealed keyword:

1. When overriding of the method of the class can cause the unexpected error.
2. When you want to prevent any third party to modify the class.

Another Important information is if we create a **static** class, it becomes automatically sealed. This means that you cannot derive a class from a static class. So, the sealed and the static class have in common that both are sealed. The difference is that you can declared a variable of a sealed class to access its members but you use the name of a static class to access its members.

Polymorphism:

polymorphism is an ability of entity to behave differently in different situation.

Polymorphism allows method to function differently based on the parameters & their data type.

Implementation:

Polymorphism in c# is implemented through method overloading & method overriding.

Abstract Classes & Interfaces:

Abstract Class: A class that is defined using abstract keyword and contain at least one method without body is called as abstract class.

Notes:

1) Abstract class can not be inistantiated using new keyword.
2) Abstract class contain atleast one abstract method & it does not contain body but for subclass it is mandatory to override (implement) abstract method.
3) Abstract class can also contain normal methods with complete implementation . Or It is not necessary that abstract class can contain only abstract methods.
4) When a method is declare as abstract in the base class then every derived class must provide its own defination for that method.
5) Abstract class can not be sealed.
6) Abstract class can also called as Half defined class.

Syntax for declaring Abstract Class:

```
  public abstract class <classname>
  {
     <access
modifier>abstract<returntype><methodname>[parameterlist];

  }
```

Where abstract specifies the class is declared as abstract class.

Implementation of Abstract class:

Abstract can implemented in the same way as base class.

Syntax :

```
Class<classname>:<abstract classname>
{
   // class members
}
```

When subclass fails to implement the abstract method then subclass can not inistantiated as compiler consider it as a abstract.

Abstract Methods:

The method that is declare in abstract class & without body are termed as abstract method.

These method are implemented in a inheriting class.

Purpose : Abstract method provides the common functionality for the classes inheriting abstract class.

Interfaces in C#:

A subclass can not inherit two or more base classes. This is because c# does not support multiple inheritance. To overcome this problem interfaces introduced.

A class in c# can implement multiple interface.

An interface is defined as a syntactical contract that all the classes inheriting the interface should follow.

Note:
1. An interface contain only abstract members
2. Unlike abstract class interface can not implement any method
3. Like abstract class interface can not inistantiated
4. An interfaces can only be inherited by classes or other interfaces.
5. In c#,all members defined in the interface have a public as a default access modifier .
6. Interfaces can not contain constants, datafields, static variables, constructor & destructor. i.e Method.

Syntax :

```
   interface<interfacename>
{
   // abstract members
}
```

Implementing Interface:

An interface implemented in a class in a similar way as of base class but you have to implement all the methods of implement. If all methods are not implemented then class can not implement. Method implemented in the class should have same signature as of declare in the interface.

Syntax:

```
          Class Classname :<InterfaceName>
     {
            //Implement interfaces
           // class members
      }
```

Multiple Interface:

Multiple inheritance implemented by a single class by using comma operator.
A class implementing multiple inheritance should has to implement all the abstract method declare in interface. But here override keyword not used as it is used in abstract class.

Syntax for multiple inheritance:

```
 class<classname> : interface1,interface2
{
    // Implement the interface method.
}
```

Explicit Interface implementation:

A class has to explicitly implement multiple inheritance if these interfaces have methods with identical names.

While implementing explicit interface you can not mention modifiers such as abstract , virtual, override or new

Syntax:

```
  class <ClassName>:<Interface1>,<Interface2>
 {
      <Access Modifier>Interface1.method();
    {
     //
    }
     <Access Modifer>Interface2.Method();
   {
   //
    }
}
```

While creating objects:

```
 Interface1 obj = new classname();
Interface2 obj2 = new classname();
```

Interface Inheritance:

An interface can inherit multiple interfaces but can not implement them.

The implementation has to done by class. And class should implement all the abstract method present in all interfaces.

Syntax:

```
Interface <InterfaceName> : <Inherited Interfacename>
{
   // Method declaration
}
```

Similarities between Abstraction & Interfaces :

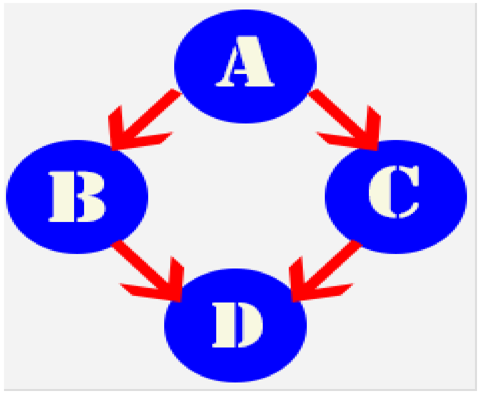
1. Neither abstract class nor interfaces are instantiated.
2. Both contain abstract methods
3. Abstract methods of the both are implemented by subclass.

Differences:

1.Abstract class can inherit the class & interfaces
Interfaces can inherit interfaces but not the class.
2.Abstract class can have method with the body .
Interfaces can not have method with the body.
3. Abstract class method is implemented using override keyword.
Interfaces method is implemented without using override keyword.
4.An abstract class is better option when you want to implement common methods & declare common abstract methods.
An interfaces is better option when you need to declare only abstract methods.
5. Abstract class can declare constructors & destructors.
An interfaces can not allowed to declare constructors & destructors.

Diamond Problem:

As you know, C# does not allow multiple class inheritance and one of the main reason behind it is Diamond Problem. You can learn better about this problem with an example. Suppose you have 4 classes named as A, B, C, and D. A is your main base class. A contains a virtual method named as PrintName. Because it is virtual method, all classes which will inherit from base class A, can override this method. B and C are 2 classes, which are inheriting from base class A and overriding its PrintName method. The class D is inheriting from both B and C, which means both the overridden methods are available for class D.
Now, there is a question. When you create an instance of D class and would try to call method PrintName, which method should be called? Method from B or C? There will be an ambiguity and this problem is called **Diamond Problem.**



Exception in C#:

Exception is a runtime error that disrupt the execution flow of instruction in programs.

The process of handling run time error are termed as Exception handling.

Example: Car stops due to technical problem, CLR sends exception as no rights to read file. etc

Types of Exception:

1. **System-Level Exception**
2. **Application-Level Exception**

System –Level Exception are thrown by system i.e. CLR
ex. Failure due to Database connectivity loss/ network connectivity loss.

Application –level exception are thrown by application.
Ex. Referencing any null object

System.Exception :

It is the base class that allows you to handle exception.

Exception Classes:

The System namespace contains the different exception classes that c# provides . The type of exception to be handled depends on the specified exception class.

Commonly Used Exception Classes:

1. System.ArithmeticException
2. System.ArgumentsException
3. System.ArrayMismatchException
4. System.DivideByZeroException
5. System.IndexOutOfRangeException
6. System.OutOfMemoryException
7. System.Data.DataException
8. System.IO.IOException
9. System.StackOverflowException
10. System.NullReferenceException

Throwing & catching Exception:

Syntax:

```
        try
           {
              // Code
            }
        catch[(Exception Class)(ObjException)]
            {
                Error handling Code.
             }
```

Where try block generates the
Generalised Catch block
Throw keyword
Finally keyword
Nested try catch
Multiple catch

Custom Exception:

Custom Exception are user defined exceptions that allow users to recognize the cause of unexpected events in a specific programs and display custom messages.

It can be created for both exception that are either thrown by CLR or UserApplication

Properties & Indexer :

In c# ,it uses a feature that allows you to set & retrives values of the field(variable) declared with any access modifier in a secure manner.
These is because properties allow you to validate values before assigning them to fields.

Uses:

Marking the class field public & exposing is a risky, as you will not have control what gets assigned & returned.
To understand this clearly with an example lets take a student class who have ID, pass mark , name. Now in this example some problem with public field
ID should not be -ve.
Name can not be set to null
Pass mark should be read only.
If student name is missing No Name should be return.

Syntax:

```
<access modifier><return type><property name>
{
  // body of the property
 }
```

Properties declaration contain special methods to read & set private values.

Property Accessor:

property accessor allows you to read & assign value to a field by implementing two special methods.

Syntax:

```
<access modifier><return type><property Name>
{
       get
       {
             // return value
       }
    set
      {
             // assign value
       }
```

// It is always mandatory to always end the get accessor with a return statement.

Types :

Properties are of mainly three types:

1. Read Only
2. Write Only
3. Read –Write

Syntax:

1. Read Only property

```
  <access modifier><return type><propertyName>
{
    get
      {
        // return value
      }
}
```

2. Write Only Property

```
 <access modifier><return type><propertyname>
{
    set
     {
       // return value
     }
}
```

Syntax:

For read write property

```
 <access modifier><return type><propertiesName>
{
   get
   {
       //return value;
   }
   set
    {
      // assign value
    }
}
```

Note: A property can be declare as static by using the static keyword. A static property is accessed using the class name & is available to entire class rather than just the instance of the class.

the get & set accessor of static property can access only static members of the class.

Indexer:

Indexer are data members that allow you to access data within objects in a way that is similar to accessing arrays.

In c# , indexers are also known as Smart Arrays.

Syntax:

```
<access modifier><return type>this[parameter list]
{
      // The get accessor.
     get
      {
         // return the value specified by index
       }
       // The set accessor.
     set
      {
         // set the value specified by index
       }
 }
```

Notes:

1.Indexer provide the faster access to data within an object as they help in indexing the data.
2.In arrays we used index position of the object to access its value. similarly an indexer allows you to used index of an object to access values within the object.
3. Implementation of indexer is similar to properties, except that the declaration of an indexer can contain parameters.
4.Indexer contain at least one parameter. parameter denotes the index position ,using which store value at that position is set or access.
5. Indexer has no name but uses this keyword

Delegates:

A delegates is a reference type variable that holds the reference to method. A reference can change at runtime.

Delegates are especially used for implementing events.
All delegates implicitly derive from System.Delegate class.

Declaring Delegates:

A delegate can refer to a method, which has same signature as that of delegate.

Syntax:

```
    <access modifier>delegate<return type>
         Delegate name [parameter list];
```

Ex. Public delegate int calculation(int a, int b);

If you declare delegates outside class then u can not declare another delegates with the same name in that namespace

Instantiating Delegate:

Syntax:

```
<delegateName><obj>=new <delegateName>(<MethodName>)
```

Ex.Calculation obj = new Calculation(Addition);

An object of the delegate is created using the new keyword
This object takes the name of the method as a parameter and these method have a same signature as that of the delegate.

The created object is used to invoke method at run time.

Why we used Delegates:

1.Encapsulating the method's call from caller
2.Effective use of delegate improves the performance of application
3.Used to call a method asynchronously

Multiple Delegate:

You can use more than one delegate in C#.

Multicast delegates:

It is a delegates which hold the reference of more than one method. Multicast delegates must contain only methods that return void , else there is run time exception.

Anonymous Methods:

We have seen , delegate are noting but the reference to method that has same signature as that of delegate.
In other words,
you can call a method that can be reference by the delegate using delegate object.

Anonymous method provide a technique to pass a code block as a delegate parameter. Anonymous methods are the methods without name having only body.
You need not specify the return type in an anonymous method; it is inferred from the return statement inside the method body.

Features: It is used in place of named method if that method is to be only invoke through delegates.

1. It appears as a inline code in the delegate declaration
2. It is best suited for small blocks
3. It can accept parameters of any type
4. Parameters using ref & out keyword can be passed to it.
5. It can include parameters of generic type.
6. It can not include jump statement such as go to , break that transfer controls out of the scope of the method.

Benefits of Anonymous Method:

1. It eliminates the need to name a method when it call only once in program.
2. It reduces number of method that needs to be maintained
3. It helps in understanding the code better where calling block & called block exist together in the same statement.

Creating Anonymous method:

It is created when we instantiated or reference a delegate with the block of code.

```
// create delegate Instance

 <access modifier>delegate<return type>delegate name(parameters);

//Instantiating delegate

<delegatename><objectdelegate>= new
    <delegatename>(parameters)
{
   /*  body of anonymous method  */
}
```

Notes:

1. When delegate keyword is used inside method body then it must be followed by anonymous method.
2. A method is defined as a set of statements while creating the object of the delegate.
3. Anonymous method are not given any return type.
4. Anonymous method are not prefixed with access modifier.

Other variables in Anonymous Method:

It can declare variables which are called outer variables. These variables are said to be captured when they get executed.

Passing parameters to Anonymous Method:

C# allows passing parameters to anonymous method. The type of parameter that can be passed to an anonymous method is specified at the time of declaring the delegate. These parameters are specified in paranthesis just like normal methods. You can pass parameters values to anonymous method while invoking the delegate.

Events:
In c#, Events allow an object (source of the event) to notify other object (subscribers) about the event(a change having occure).

An event is a user-generated or system generated action that enables the required objects to notify other objects or classes to handle the event.

An event are actions that can trigger methods to execute a set of statements.

Features :
1.It can be declare in classes or interfaces
2.It can be declare as abstract or sealed
3.It can be declare as virtual
4.They are implementing using delegates.

Use: Events can be used to perform the customized action that are not already supported by c#.
Events in c# is mostly used in GUI based application where event such as selecting item at from a list & closing a window are tracked.

Creating & Using Events:

Delegates are widely used for event handling.

Declaring Events :

It consist of two steps ,creating delegate and creating event

Declaring Delegate:

<access modifier>delegate<return type><Identifier> (parameters);

Declaring Event:

<access modifier>event<delegate><Event name>;

Subscribing to Event:

An object can subscribe to event only if the event exists.To subscribe to the event object adds a delegates that calls a method when the event is raised. This takes place by associating the event handler to the created event using the +=addition assignment operator. These is known as subscribing to event.
To unsubscribe from the event use -=sub assignment operator.

Syntax to create a method in receiver class:

<access modifier><return type><methodname>(parameters);

Syntax to associate the method with the event:

<objname>.<eventname>+=new <delegatename>(methodname);

Where objname is the object of the class in which event handler is defined.

Raising an Event:

The class containing the event is used to publish the event.This is called as publishers class.
Some other class that accept these events is called as subscriber class.

Collection in C#:

Collection classes allows you to control & manipulate group of object at run time.

The System.Collections namespace consist of collection arrayslists,hashtables, dictionaries etc.

The System.Collection.Generic namespace consist of generic collection, which provide better type safety and performance.

Example of collection: Shoping Cart

A set of values either of same types or different types & size of the set may not known initially. The values in the collection can be inserted or modified at runtime.

Def: A collection is a set of related data that may not necessarily belongs to the same data type. It can be set or modified dynamically at run time.

Accessing collection is similar to accessing array ,where elements are accessed by their index numbers.
But there are difference between Array & Collection

Array:
1. Can not resize at run time
2. The individual element are of same type.
3. Do not contain any methods for operation on elements

Collection:
1. Can be resized at runtime
2. Individual element may or may not be of same type.
3. Contain methods for operation on elements.

System.Collections Namespace:

It consist of various collection such as
1. Dynamic arrays
2. List
3. Dictionaries

Commonly Used classes & interfaces in system.Collection Namespace:

1.ArrayList: Same as array except item can added or removed at run time & can be different type .

2.Stack: Follows LIFO i.e. Last In First Out

3.Queue: Follows FIFO i.e. First In First Out

4.HashTable : Provides collection of key & value pairs that are arranged based on the hash code of the key.

5.SortedList: Provides the collection of key & value pairs where item are sorted based on the keys.

Interfaces:
1.ICollection:specifies the size & synchronization methods for all collection
2.IEnumarator: Supports iteration over the element of the collection
3.Ilist: Represents a collection of items that can be accessed by their index number.

How to use Array List Class:

How to add an Item in an ArrayList ?

```
Syntax : ArrayList.add(object)
object : The Item to be add the ArrayList
ArrayList arr;

arr.Add("Item1");
```

How to Insert an Item in an ArrayList ?

```
Syntax : ArrayList.insert(index,object)
index : The position of the item in an ArrayList
object : The Item to be add the ArrayList
ArrayList arr;

arr.Insert(3, "Item3");
```

How to remove an item from arrayList ?

```
Syntax : ArrayList.Remove(object)
object : The Item to be add the ArrayList

arr.Remove("item2")
```

How to remove an item in a specified position ArrayList ?

```
Syntax : ArrayList.RemoveAt(index)
index : the position of an item to remove from an ArrayList

ItemList.RemoveAt(2)
```

How to sort ArrayList ?

```
Syntax : ArrayList.Sort()
```

Hash Table :

In C#, Hash table allows you to store value in the form of Key & values. It generates hash table which associates keys with their corresponding values .The Hash table class uses the hash table to retrives value associated with their unique key**.**

Ex. Reception area of hotel having keyholder storing the bunch of key

Hash table class uses hashing technique to retrive the corressponding value of the key.
Hashing is a process of generating hash code for the key. That code is used to identify corresponding value of the key.

How to use Hash Table :

Add : To add a pair of value in HashTable

```
Syntax : HashTable.Add(Key,Value)
Key : The Key value
Value : The value of corresponding key
Hashtable ht;

ht.Add("1", "Sunday");
```

ContainsKey : Check if a specified key exist or not

```
Synatx : bool HashTable.ContainsKey(key)
Key           : The Key value for search in HahTable
Returns : return true if item exist else false

ht.Contains("1");
```

ContainsValue : Check the specified Value exist in HashTable

```
Synatx : bool HashTable.ContainsValue(Value)
Value : Search the specified Value in HashTable
Returns : return true if item exist else false

ht.ContainsValue("Sunday")
```

Remove : Remove the specified Key and corresponding Value

```
Syntax : HashTable.Remove(Key)
Key : The key of the element to remove

ht.Remove("1");
```

Sorted List:

It represent the collection of key & value pairs where element are sorted according to key.

By default sorted class sort in asending order.

These can be changed if an Icomparable object is passed to the constructor of the sorted list class. These elements are accessed using the corresponding keys or index numbers.

If you access element using their keys, sorted list class behaves as hash table and if you access element using their index , it behave like array list.

Sorted List is nothing but the combination of Array List & Hash Table.

Difference between Hashtable Class & SortedList Class is that values in the object of the sortedlist class are sorted by their keys & accessible by their keys & indexer . It leads to program execution slower than Hashtable.

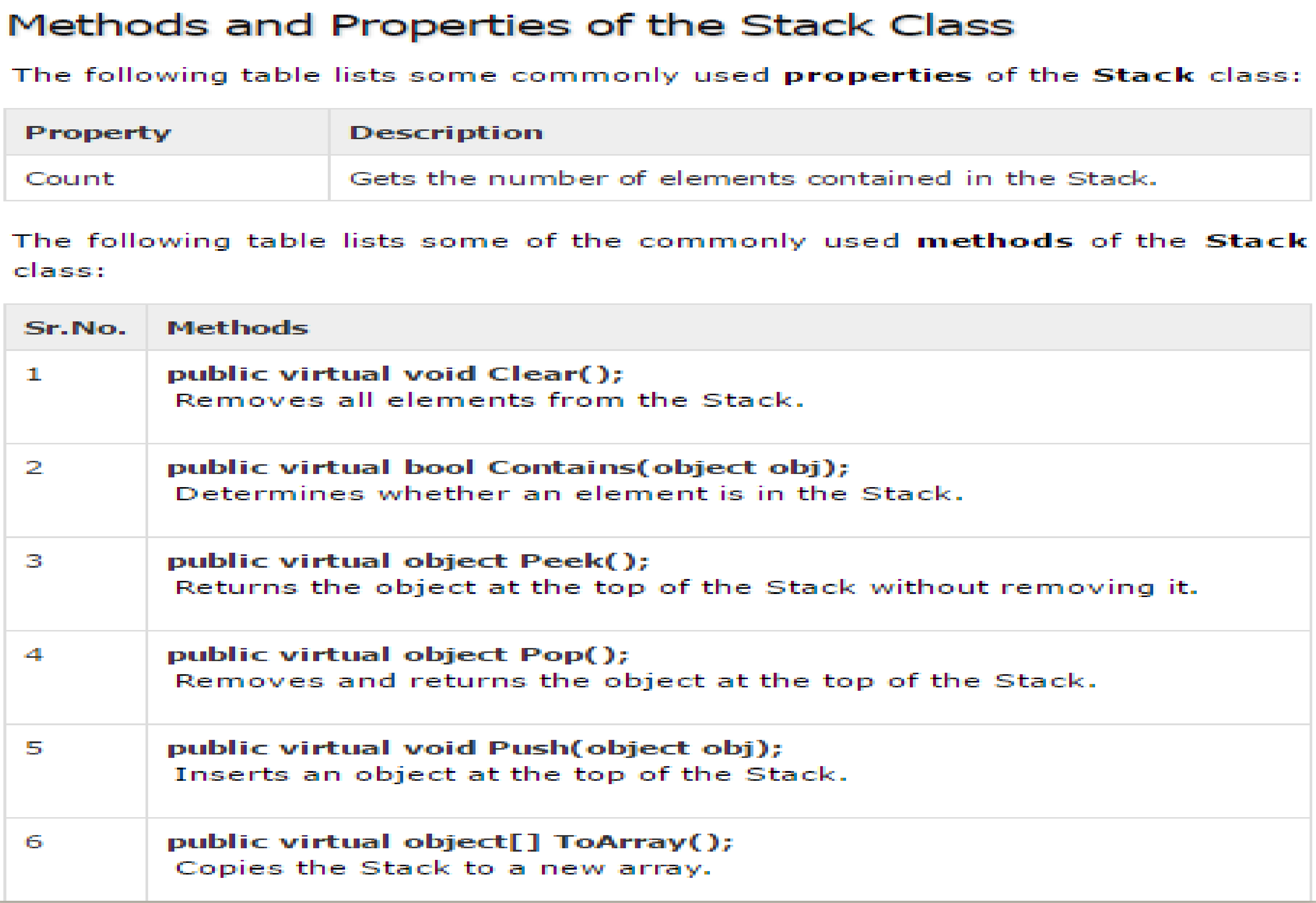
Notes:

C# has generic and non-generic SortedList.

SortedList stores the key-value pairs in ascending order of the key. Key must be unique and cannot be null whereas value can be null or duplicate.

Non-generic SortedList stores keys and values of any data types. So values needs to be cast to appropriate data type

Stack Class: Last In First Out or First In Last Out



Methods and Properties of the Stack Class

The following table lists some commonly used **properties** of the **Stack** class:

Property	Description
Count	Gets the number of elements contained in the Stack.

The following table lists some of the commonly used **methods** of the **Stack** class:

Sr.No.	Methods
1	**public virtual void Clear();** Removes all elements from the Stack.
2	**public virtual bool Contains(object obj);** Determines whether an element is in the Stack.
3	**public virtual object Peek();** Returns the object at the top of the Stack without removing it.
4	**public virtual object Pop();** Removes and returns the object at the top of the Stack.
5	**public virtual void Push(object obj);** Inserts an object at the top of the Stack.
6	**public virtual object[] ToArray();** Copies the Stack to a new array.

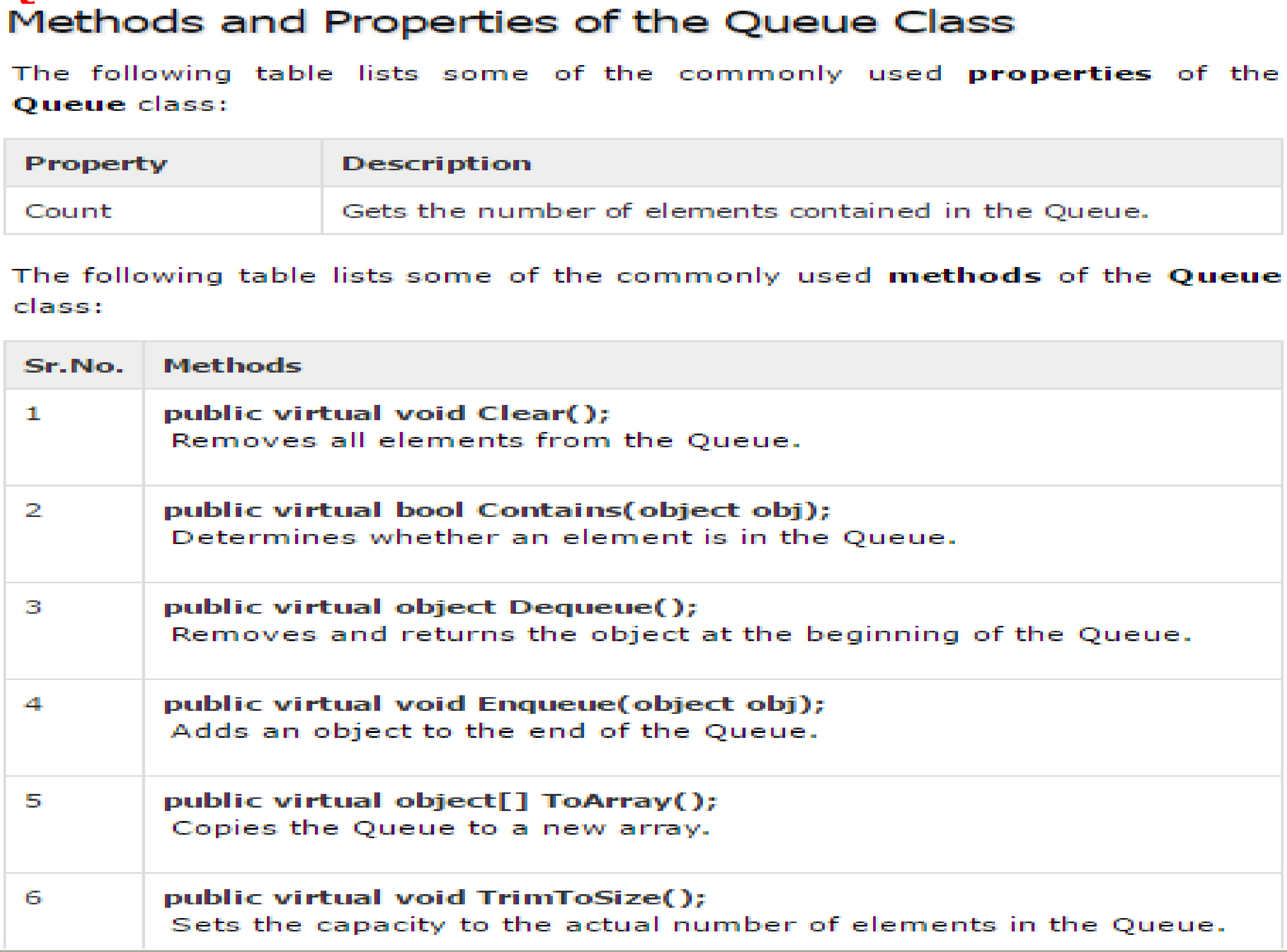
Notes for Stack:

1. Stack stores the values in LIFO (Last in First out) style.
2. The element which is added last will be the element to come out first.
3. Use the Push() method to add elements into Stack.
4. The Pop() method returns and removes elements from the top of the Stack.
5. Calling the Pop() method on the empty Stack will throw an exception.
6. The Peek() method always returns top most element in the Stack

Notes for Queue:

1. The Queue stores the values in FIFO (First in First out) style.
2. The element which is added first will come out First.
3. Use the Enqueue() method to add elements into Queue
4. The Dequeue() method returns and removes elements from the beginning of the Queue.
5. Calling the Dequeue() method on an empty queue will throw an exception.
6. The Peek() method always returns top most element

Queue Class: FIFO or LILO



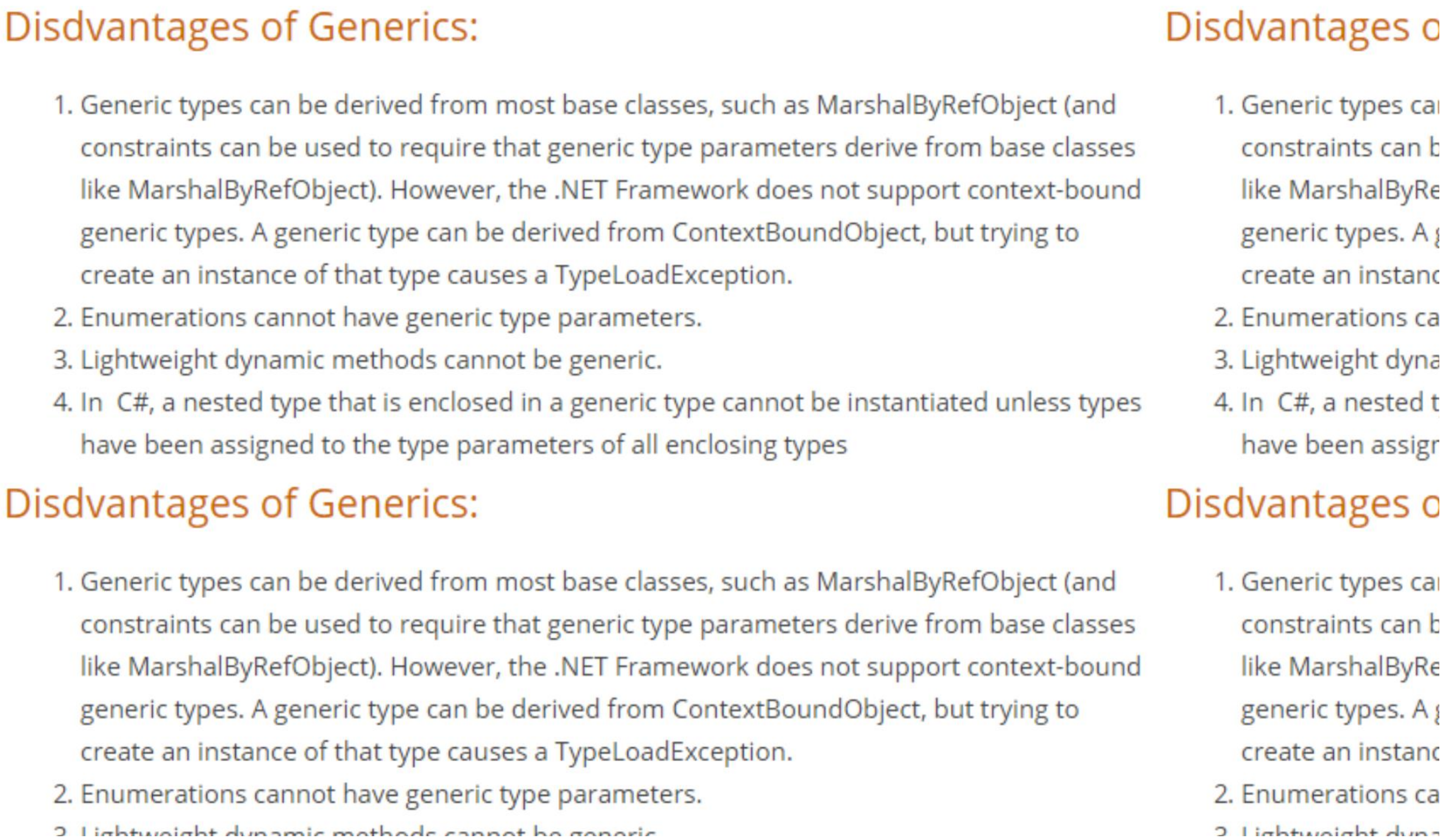
Methods and Properties of the Queue Class

The following table lists some of the commonly used **properties** of the **Queue** class:

Property	Description
Count	Gets the number of elements contained in the Queue.

The following table lists some of the commonly used **methods** of the **Queue** class:

Sr.No.	Methods
1	**public virtual void Clear();** Removes all elements from the Queue.
2	**public virtual bool Contains(object obj);** Determines whether an element is in the Queue.
3	**public virtual object Dequeue();** Removes and returns the object at the beginning of the Queue.
4	**public virtual void Enqueue(object obj);** Adds an object to the end of the Queue.
5	**public virtual object[] ToArray();** Copies the Queue to a new array.
6	**public virtual void TrimToSize();** Sets the capacity to the actual number of elements in the Queue.



Disdvantages of Generics:

1. Generic types can be derived from most base classes, such as MarshalByRefObject (and constraints can be used to require that generic type parameters derive from base classes like MarshalByRefObject). However, the .NET Framework does not support context-bound generic types. A generic type can be derived from ContextBoundObject, but trying to create an instance of that type causes a TypeLoadException.
2. Enumerations cannot have generic type parameters.
3. Lightweight dynamic methods cannot be generic.
4. In C#, a nested type that is enclosed in a generic type cannot be instantiated unless types have been assigned to the type parameters of all enclosing types

Disdvantages of Generics:

1. Generic types can be derived from most base classes, such as MarshalByRefObject (and constraints can be used to require that generic type parameters derive from base classes like MarshalByRefObject). However, the .NET Framework does not support context-bound generic types. A generic type can be derived from ContextBoundObject, but trying to create an instance of that type causes a TypeLoadException.
2. Enumerations cannot have generic type parameters.

Regular Expression:

A regular expression is a pattern that could be match against an input text. The .Net framework provides a regular expression engine that allows such matching . A pattern consist of one or more character literals, operators or constructs.

Lots of construct for defining Regular Expression:

1. Character escapes
2. Character classes
3. Anchors
4. Grouping constructs
5. Quantifiers
6. Backreference constructs
7. Alternation constructs
8. Substitutions
9. Miscellaneous constructs

Regex Class:

A Regex class is used for representing a regular expression. It has following commonly methods

Sr.No	Methods
1	**public bool IsMatch(string input)** Indicates whether the regular expression specified in the Regex constructor finds a match in a specified input string.
2	**public bool IsMatch(string input, int startat)** Indicates whether the regular expression specified in the Regex constructor finds a match in the specified input string, beginning at the specified starting position in the string.
3	**public static bool IsMatch(string input, string pattern)** Indicates whether the specified regular expression finds a match in the specified input string.
4	**public MatchCollection Matches(string input)** Searches the specified input string for all occurrences of a regular expression.
5	**public string Replace(string input, string replacement)** In a specified input string, replaces all strings that match a regular expression pattern with a specified replacement string.
6	**public string[] Split(string input)** Splits an input string into an array of substrings at the positions defined by a regular expression pattern specified in the Regex constructor.

Generics:

Generics are data structures that allow you to reuse the same code for different types such as classes, interfaces & so forth.

Generics mostly used while working with arrays & enumarator collections.

Iterators are the block of the code that can iterate through the values of the collection.

Purpose of Generics :

Example of Branded & general T-shirt.

Similar to general t shirt , in C# generics are data structures that allow you to reuse the defined functionalities for different c# data types.

Definition of Generic:

Example. Storing names of students from console and boxed it and store as object type.

i.e. type safety avoided,

But generics provide the type safety, which has no of feature including the ability to allow you to define generalised type template based on which type can be constructed later.

Generics are a kind of the structure that can work with value types as well as reference types.

The System.Collections.Generic namespace consists of classes & interfaces that allow you to define customized generic collections.

Creating Generics:

A generic declaration always accepts a type parameter, which is a placeholder for the required data type.
A type is specified only when a generic type is referred to or constructed as a type within a program.

Syntax:
<Classname,MethodName or InterfaceName><<T>>;

Benefits:
Generic ensure type safety at compile time . Generic allows you to reuse code in safe manner without casting or boxing. A generic type is reusable with different types but can accept values of a single type at a time.

Creating Generic Classes:

Generic classes defined the functionality that can be used for any data type.
Generic classes are declared with a class declaration followed by type parameter enclosed within angular brackets.

While declaring a generic class you can apply some restriction or constraints to the type parameter by using where keyword . However applying constraint to a type parameter is optional.

Thus while creating a generic class you must generalise a data type into the type parameters .

Syntax:

```
  <access modifier>class <classname><<type parameter list>>
 [where <type parameter constraint class>]
```

Note: Generic classes can be nested within other generic or non generic classes .However any class nested within generic class itself a generic class since type parameter of outer class is supplied to inner class.

Generic classes with Constraint on Parameters:

Inheriting Generic Classes:

A generic class can inherited like general class hence generic class act as a base class & also derived class.

While inheriting a generic class in another generic class , you can use the generic type parameter of the base class instead of passing datatype of the parameter

Generic to Generic

```
Public class student<T>
{

}
Public class mark<T>:student<T>
{

}
```

Generic to Non Generic: while inheriting generic class in non generic class, you must provide the data type of the parameter instead of the Base class generic type parameter.

Generic to Non-generic

```
Public class student<T>
{
}
Public class mark:student<int>
{

}
```

Generic Method: Generic method process values whose datatypes are known only when accessing the variables that store these values. A generic method is declared with the generic type parameter list enclosed with angular brackets.

You can declare the generic method inside generic or non generic class.

Generic method can be declare with

1. Virtual
2. Override
3. Abstract

The generic method declare with the virtual keyword can be overridden in derived class.

The generic method declare with abstract keyword contains only declaration & does not contain body & such a method is implemented in derived class.

Generic Interfaces:

Generic Interface Constrain :

Generic Interface:

Generic interfaces are useful for generic collections or generic classes representing items in the collection. You can use generic classes & interfaces to avoid boxing & unboxing operation on value types .

Dictionary Generic Class:

It consist of generic collection of elements organized in Key & Value pairs.
It maps keys to their corresponding values .

Syntax:

Dictionary<Tkey , Tvalue>

Where Tkey is the type parameter of the keys to be store in the instance of dictionary class.

Dictionary class does not allow null values as a element. The capacity of Dictionary class is the number of elements that it can hold. However elements are added ,the capacity is automatically increased.

Link List :

Difference Between Generic collection & non generic collection:
i.e. arraylist & list
1. ArrayList belongs to the System.Collections namespace.
2. ArrayList does not have type restriction for storing data i.e. it is not Type Safe. You can store anything in ArrayList. In fact same ArrayList can store multiple types of objects.
3. ArrayList stores all data as object thus to get it back you must remember what you stored where and correspondingly Type Cast it as per its original Type when stored
4. While running a Loop on ArrayList you need to use Object data type. Thus this is another disadvantage as you again do not know what type of data particular item contains.

1. Generic List (List<T>) belongs to the System.Collections.Generic namespace,
2. In Generic List (List<T>), T means data type, i.e. string, int, DateTime, etc. Thus it will store only specific types of objects based on what data type has been specified while declarations i.e. it is Type Safe. Thus if you have a Generic List of string you can only store string values, anything else will give compilation error.
3. Generic List stores all data of the data type it is declared thus to getting the data back is hassle free and no type conversions required
4. While running a Loop on Generic List again it is problem free as we

Multithreading In C# :

What is Process: Process is what the operating system uses to facilitate the execution of a program by providing the required resources. Each process has unique id associated with it. You can view the process within which a program is being executed using windows task manager.

What is Thread: Thread is a light weight process . A process has atleast one thread which is commonly called as main thread which actually executes the application code .
A single process can have multiple thread.
Note: All threading related classes are present in System.Threading Namespace.
What is MultiThreading :

C# supports parallel execution of job or program code through multithreading. Multithreading contains two or more program codes which run concurrently without affecting to each other and each program codes is called thread.

Multithreading is nothing but the two or more threads run concurrently /simulteneously without affecting to each other.

Multithreading is a process of executing multiple threads simultaneously.

Thread is basically a lightweight subprocess, a smallest unit of processing.
Multiprocessing and multithreading, both are used to achieve multitasking. But we use multithreading than mulitprocessing because threads share a common memory area.

They don't allocate separate memory area so save memory, and context-switching between the threads takes less time than processes.

Multithreading is mostly used in games, animation etc

Multitasking:

Multitasking is a process of executing multiple task simultneously. We use multitasking to used CPU memory. It is done in two ways.

1.Process based Multitasking(Multiprocessing)

2.Thread based Multitasking(Multithreading)

❑Process-based Multitasking (Multiprocessing)

- Each process have its own address in memory i.e. each process allocates separate memory area.
- Process is heavyweight.
- Cost of communication between the process is high.
- Switching from one process to another require some time for saving and loading registers, memory maps, updating lists etc.

❑Thread-based Multitasking (Multithreading)

- Threads share the same address space.
- Thread is lightweight.
- Cost of communication between the thread is low.
- **Note:** At least one process is required for each thread.

App Domain:

An application domain is a segmentation within a process; it provides a level of isolation and security within each process.

Threads are actually associated with a single AppDomain and not a Process.

The CLR creates a default AppDomain and its default Thread as part of executing a .NET assembly.

A Process can contain multiple AppDomains and AppDomains can contain multiple Threads; however these will only exist if if they are created manually by executing code within the Process.

AppDomains are less expensive than Processes; the CLR can load or unload AppDomains quicker than a processes can be created or destroyed.

When an Assembly is loaded it is loaded into an AppDomain. Code within an assembly can only be executed within the AppDomain it was loaded into.

An Application Domain is a light-weight process that functions as a unit of isolation within that process to further function as a logical container that allows multiple assemblies to run within a single process which prevents them from directly accessing other assemblies’ memories. While considered a light-weight process, AppDomains still offer many of the features that a Windows process offers: separate memory spaces and access to resources. AppDomains are actually more efficient than a process by enabling multiple assemblies to be run in separate application domains without the overhead of launching separate processes.

The GetDomain() method returns the name of executable file in which the thread is executing. This is a static method so we use it with the name of class.

Types to create thread:

There are two types to create thread in C#, first one is through the Thread Class and second one is through the Thread Start Delegate.

With the help of thread we can increase the response time of the application. While working with desktop application development, we may sometimes required to work with Multithreaded programming. It can also help in optimizing the performance of the software as it make most usage of the CPU core(s) and thus providing better throughput/results. We refer to this type of implementation as 'multi-tasking' also. As in such case, our application will be able to do different type of tasks at the same time, in parallel

Thread Priorities:

It specifies the scheduling priority of the thread and executions performs on these priority.

When we create any thread and not assign its priority then by default normal priority will be assign for it. The thread priority does not affect the state of thread.

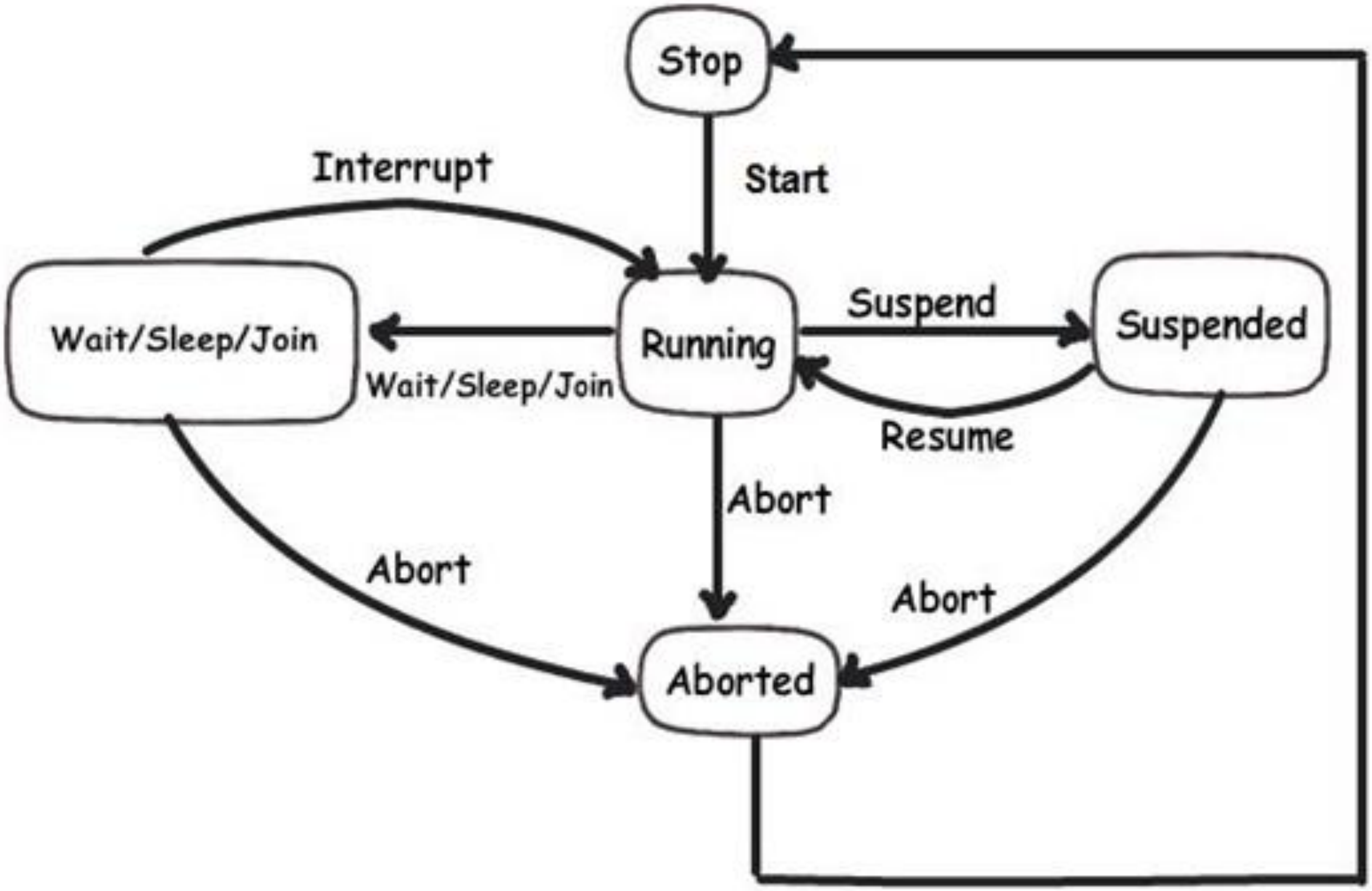
Foreground Thread:

Even if app exit foreground thread is running.

Background thread:

It quits when your main application exit.

Life Cycle of C# threads: 4 states WARS



Thread life cycle in C# application

The life cycle of a thread starts when an object of the System.Threading.Thread class is created. The life cycle of The thread ends with task execution. There are various states in the life cycle of a thread.

These states are:

- **The Unstarted state:** When an instance of the Thread class is created, the thread enters the unstarted state. a new thread is an empty object of the **thread**class, and no system resources such as memory are allocated to it. You have to invoke the **Start ( )** method to start the thread.

- **The Runnable state:** The thread remains in the unstarted state until the program call the **Start ( )** method of the **Thread** class, which places the thread in the Runnable state and immediately returns control to the calling thread. This state is also called as the **ready** or **started state.** The newly started thread and any other thread in the program execute concurrently. A single processor cannot execute more than one thread at a time.

- **The Not Runnable state:**
  A thread is not in the Runnable state if it is:

1. Sleeping: A thread is put into the sleeping mode by calling the **Sleep ( )** method. A sleeping thread enters the Runnable state after the specified time of sleep has elapsed.

2. Waiting: A thread can be made to wait for some specified condition to be satisfied by calling the **Wait ( )** method. The thread can be notified of the condition by invoking the **Pulse ( )** method of the **Thread** class.

3. Blocked: A thread could be blocked by an I/O operation. When the thread is blocked, it enters the not Runnable state.

- **The Dead state:** A running thread enters the dead state when the statements of the thread method are complete. This state is also called the **terminated state**. A program can force a thread into the dead state by calling the **Abort ( )** method of the **Thread** class on the appropriate thread object. The **Abort ( )** method throws a **ThreadAbortException** in the thread, normally causing the thread to terminate. When a thread is in the dead state and there are no references to the thread object, the garbage collector can remove the thread object from memory.

Foreground Thread & background Thread:

The execution of a C# program starts with a single thread called the main thread that is automatically run by the CLR and Operating System.

From the main thread, we can create other threads for doing the desired task in the program. The process of developing a program for execution with multiple threads is called multithreaded programming and the process of execution is called multithreading.

In C# there're the following 2 kinds of threads.
Foreground Thread
Background Thread
1. Foreground Thread
Foreground threads are those threads that keep running even after the application exits or quits.
It has the ability to prevent the current application from terminating.
The CLR will not shut down the application until all Foreground Threads have stopped.

2. Background Thread
Background Threads are those threads that will quit if our main application quits. In short, if our main application quits, the background thread will also quit.
Background threads are views by the CLR and if all foreground threads have terminated, any and all background threads are automatically stopped when the application quits.
By default every thread we create is a Foreground Thread.

Let's look at an example.

In this example we'll create 2 Threads and will run those threads in parallell and we'll also see how Foreground and Background threads work.

LINQ:

LINQ stands for Language Integrated Queries. Which is descriptive for where it is used & what it does. The LIN part means that it is a part of programming language syntax & other part Q means quering data.

Finally LINQ is a programming language syntax that is used to query data.

Why we need LINQ:

Language Integarted Query(**LINQ**) was introduced in **C# 3.0** & .NET framework **3.5**, the idea behind this was, suppose some day we are working on SQL server database so we need to learn *SQL server*syntax, Sql related ADO.NET objects. Now suppose after some time we switched to a *Oracle database*, so again we need to garb command on oracle related syntax & oracle related ADO.NET objects to work with C#, so basically this problem arises with the number of technologies we adapt, we need to learn all of them. So basically to avoid this, Microsoft introduced LINQ where we can not only work with collections populated from database, but with various other data sources as well like*XML* file, a in-memory datatable etc. We can even use LINQ when working with *Reflections*. In .Net any datasource which implements IEnumerable interface can be used for querying, filtering, grouping, ordering & projecting data.

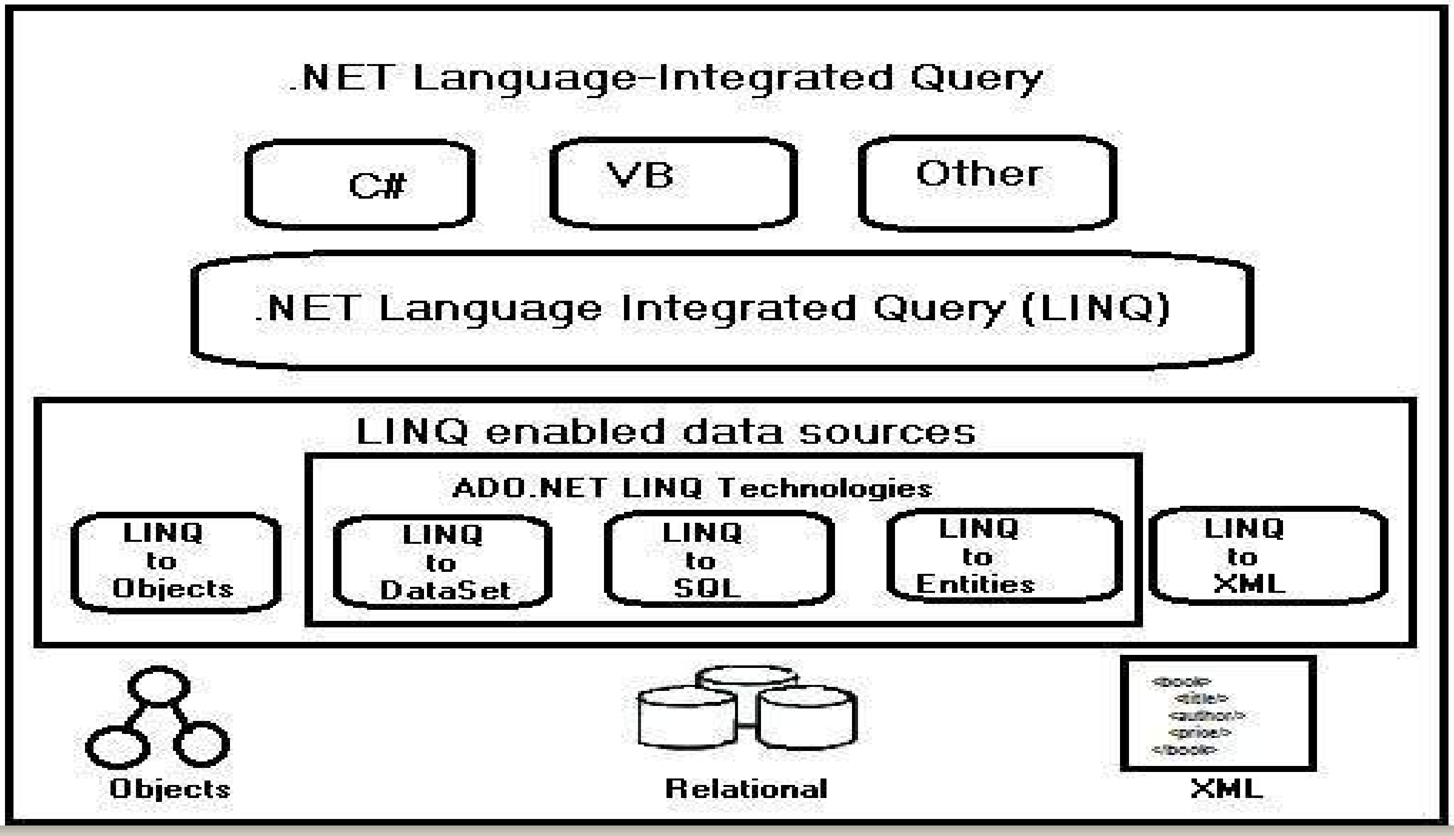
Types of LINQ:

1. LINQ to Objects
2. LINQ to XML(XLINQ)
3. LINQ to dataset
4. LINQ to SQL
5. LINQ to Entities

LINQ Architecture:

LINQ Architecture in .NET

LINQ has a 3-layered architecture in which the uppermost layer consists of the language extensions and the bottom layer consists of data sources that are typically objects implementing IEnumerable<T> or IQueryable<T> generic interfaces. The architecture is shown below in Figure.


.NET Language-Integrated Query
C#
VB
Other
.NET Language Integrated Query (LINQ)
LINQ enabled data sources
ADO.NET LINQ Technologies
LINQ to Objects
LINQ to DataSet
LINQ to SQL
LINQ to Entities
LINQ to XML
Objects
Relational
XML

Query Expression:

It is nothing but the LINQ query, expressed in a form that is similar to that of SQL with query operators like select ,where, and order by. Query Expression usually start with the keyword "From".

Advantages of LINQ:

1. Because LINQ is integrated into the C# language, it provides syntax highlighting and IntelliSense. These features make it easy to write accurate queries and to discover mistakes at design time.
2. Because LINQ queries are integrated into the C# language, it is possible for you to write code much faster than if you were writing old style queries. In some cases, developers have seen their development time cut in half.
3.Because LINQ is extensible, you can use your knowledge of LINQ to make new types of data sources queriable.
4. After creating or discovering a new LINQ provider, you can leverage your knowledge of LINQ to quickly understand how to write queries against these new data sources.
5.The integration of queries into the C# language also makes it easy for you to step through your queries with the integrated debugger.

Disadvantage:

1. There are no clear outlines in LINQ application.
2. Its complex to understand the advance query using expressions.
3. Joins are very slow when used in query.

LINQ standards Query Operator:

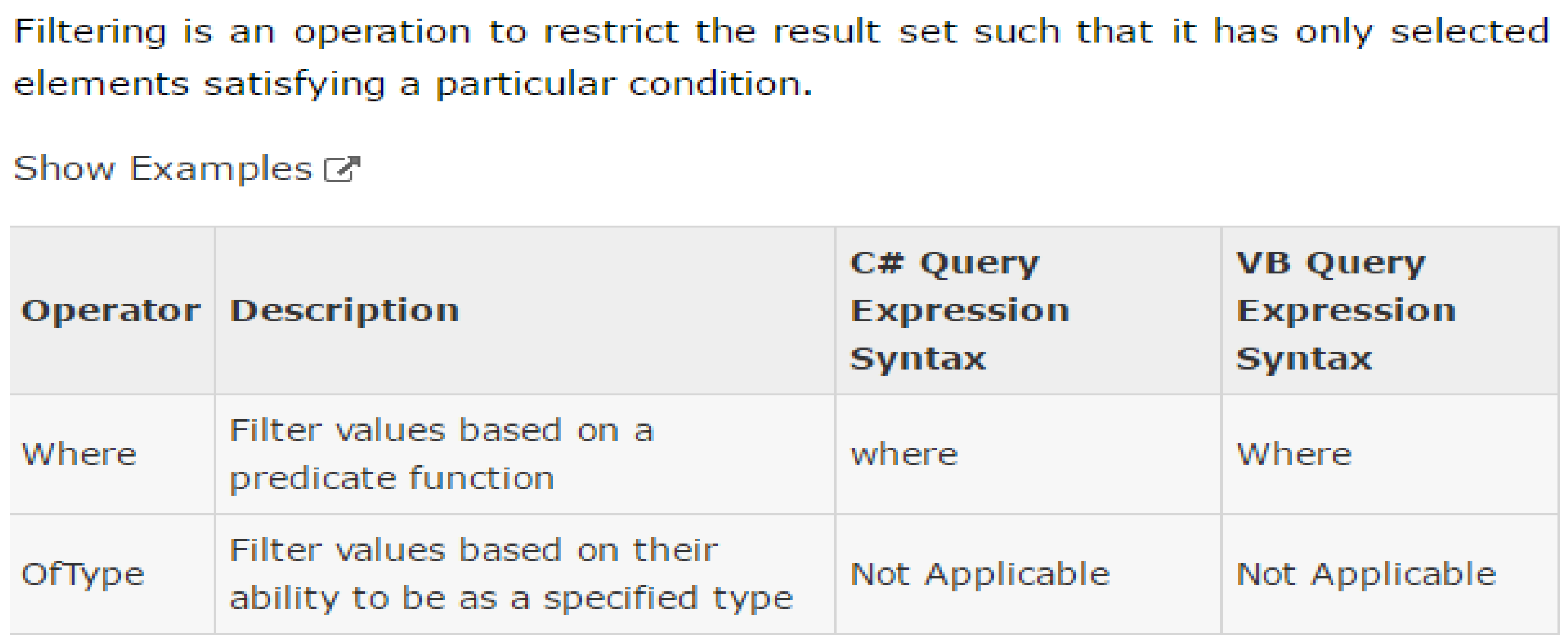
A set of extension method forming a query pattern is known as Query Operators.

LINQ standard operator are categorized on the basis of their functionality

1.Filtering Operators
2.Join Operators
3.Projection Operations
4.Sorting Operators
5.Grouping Operators
6.Conversions
7.Concatenation
8.Aggregation
9.Quantifier Operations
10.Partition Operations

11.Generation Operations
12.Set Operations
13.Equality
14.Element Operators

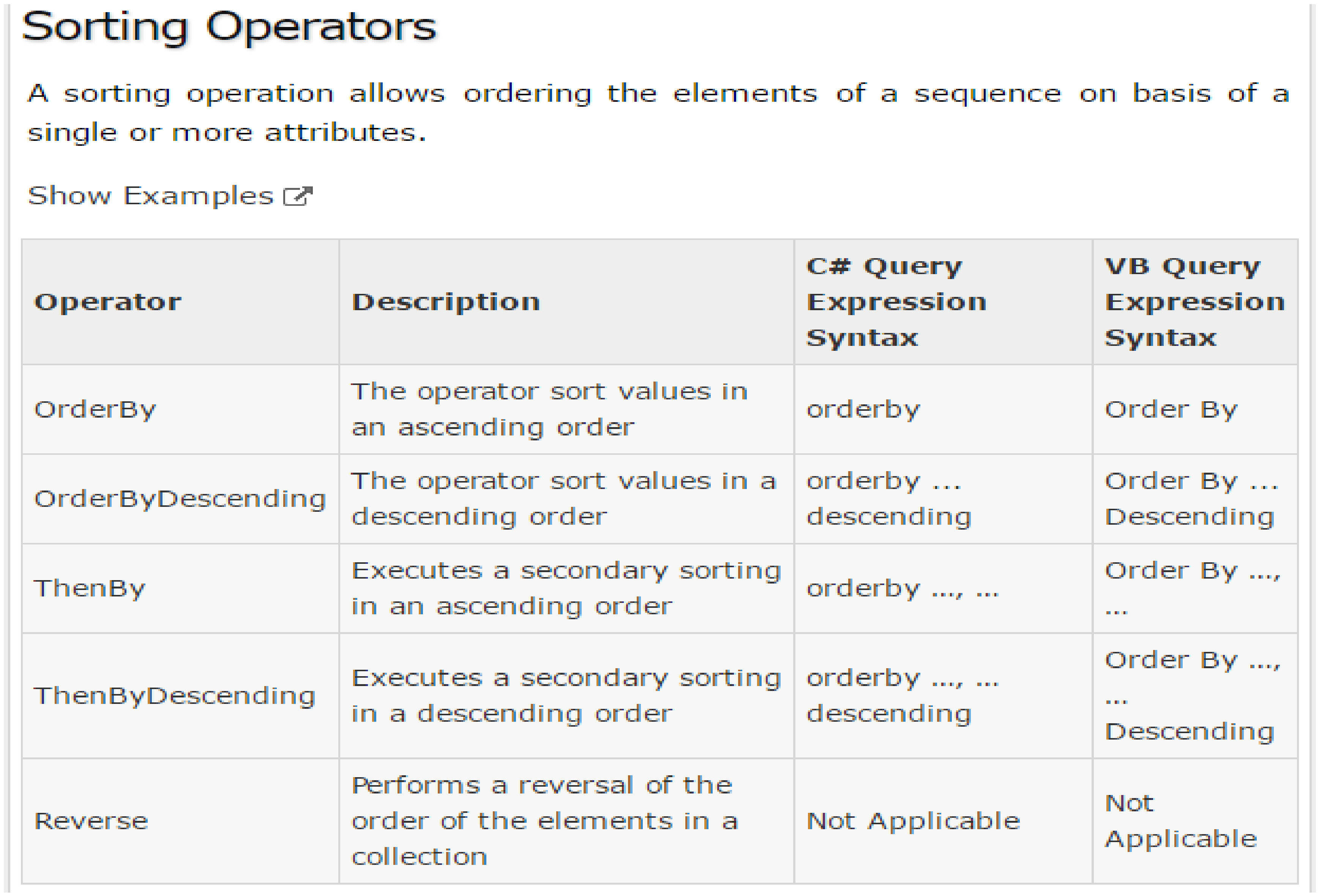
Filtering Operators:
Filtering is an operation to restrict the result set such that it has only selected elements satisfying a particular condition.



Filtering is an operation to restrict the result set such that it has only selected elements satisfying a particular condition.

Show Examples

Operator	Description	C# Query Expression Syntax	VB Query Expression Syntax
Where	Filter values based on a predicate function	where	Where
OfType	Filter values based on their ability to be as a specified type	Not Applicable	Not Applicable



Sorting Operators

A sorting operation allows ordering the elements of a sequence on basis of a single or more attributes.

Show Examples

Operator	Description	C# Query Expression Syntax	VB Query Expression Syntax
OrderBy	The operator sort values in an ascending order	orderby	Order By
OrderByDescending	The operator sort values in a descending order	orderby ... descending	Order By ... Descending
ThenBy	Executes a secondary sorting in an ascending order	orderby …, …	Order By …, …
ThenByDescending	Executes a secondary sorting in a descending order	orderby …, … descending	Order By …, … Descending
Reverse	Performs a reversal of the order of the elements in a collection	Not Applicable	Not Applicable

Grouping Operators

The operators put data into some groups based on a common shared attribute.

Show Examples

Operator	Description	C# Query Expression Syntax	VB Query Expression Syntax
GroupBy	Organize a sequence of items in groups and return them as an IEnumerable collection of type IGrouping<key, element>	group … by -or- group … by … into …	Group … By … Into …
ToLookup	Execute a grouping operation in which a sequence of key pairs are returned	Not Applicable	Not Applicable

Lambda Expression:

The term ‘Lambda expression’ has derived its name from ‘lambda’ calculus which in turn is a mathematical notation applied for defining functions. Lambda expressions as a LINQ equation’s executable part translate logic in a way at run time so it can pass on to the data source conveniently.

However, lambda expressions are not just limited to find application in LINQ only.

These expressions are expressed by the following **syntax:**

(input parameters) => expression or statement block

Below is an example of lambda expression
y => y * y

Expression Lambda:

As the expression in the syntax of lambda expression shown above is on the right hand side, these are also known as expression lambda.

Expression Tree :

Lambda expressions are used in **Expression Tree** construction extensively. An expression tree give away code in a data structure resembling a tree in which every node is itself an expression like a method call or can be a binary operation like x<y. Below is an example of usage of lambda expression for constructing an expression tree.

Statement Lambda:

There is also **statement lambdas** consisting of two or three statements, but are not used in construction of expression trees. A return statement must be written in a statement lambda.

Syntax of statement lambda

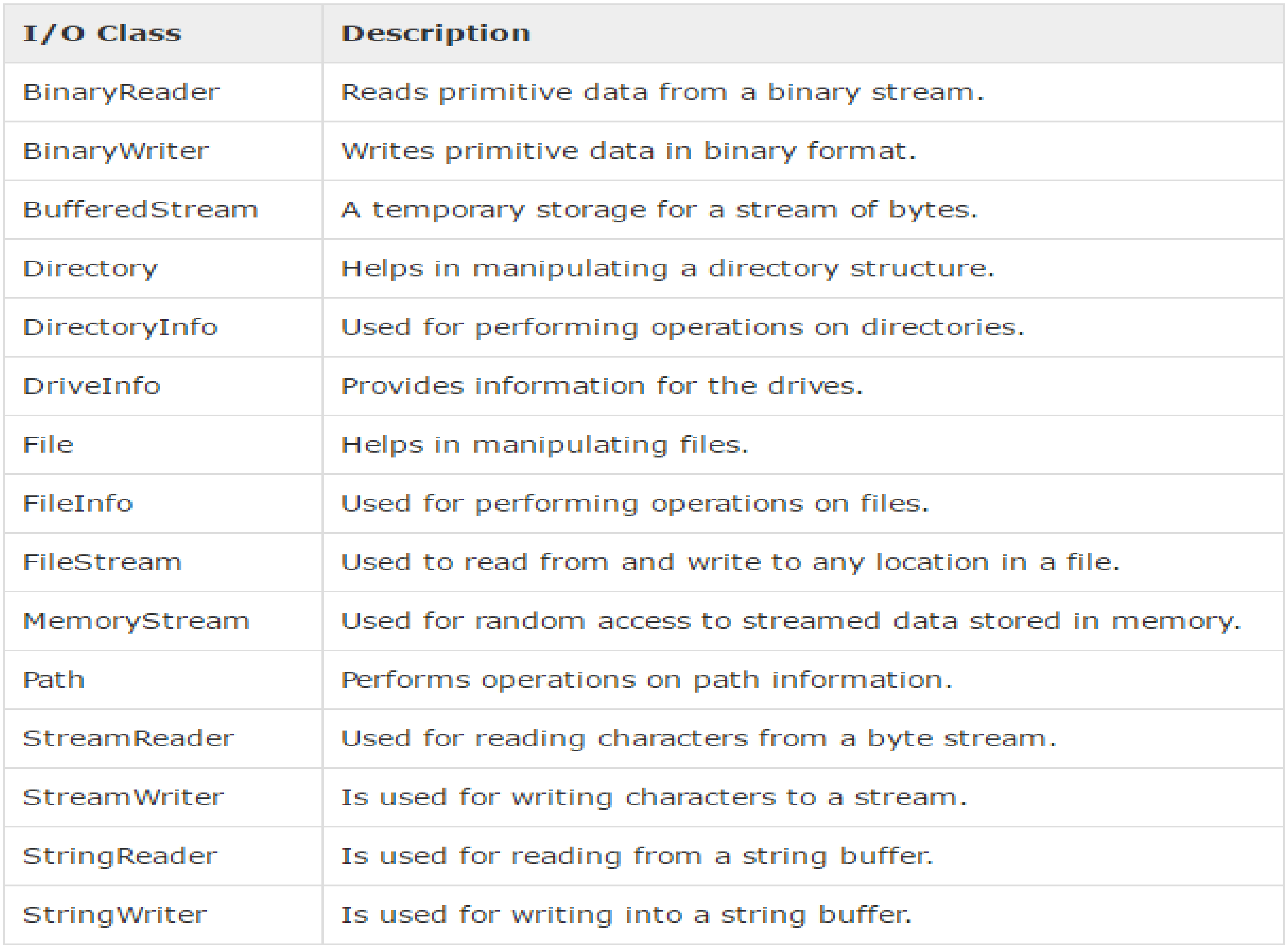
```
(params) => {statements}
```

FILE HANDLING:

A **file** is a collection of data stored in a disk with a specific name and a directory path. When a file is opened for reading or writing, it becomes a **stream**.

The stream is basically the sequence of bytes passing through the communication path. There are two main streams: the **input stream** and the**output stream**. The **input stream** is used for reading data from file (read operation) and the **output stream** is used for writing into the file (write operation).

The System.IO namespace has various classes that are used for performing numerous operations with files, such as creating and deleting files, reading from or writing to a file, closing a file etc.



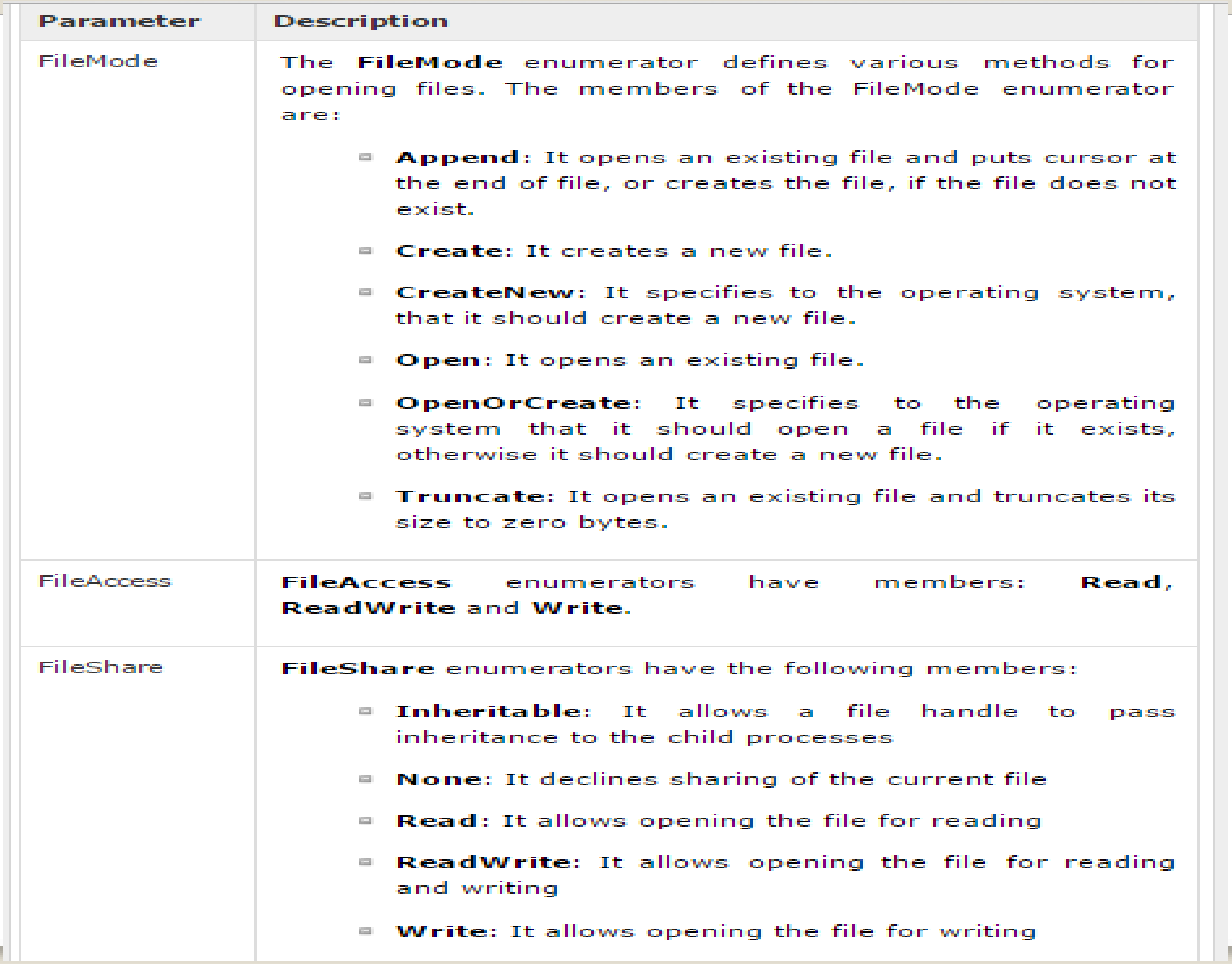
I/O Class	Description
BinaryReader	Reads primitive data from a binary stream.
BinaryWriter	Writes primitive data in binary format.
BufferedStream	A temporary storage for a stream of bytes.
Directory	Helps in manipulating a directory structure.
DirectoryInfo	Used for performing operations on directories.
DriveInfo	Provides information for the drives.
File	Helps in manipulating files.
FileInfo	Used for performing operations on files.
FileStream	Used to read from and write to any location in a file.
MemoryStream	Used for random access to streamed data stored in memory.
Path	Performs operations on path information.
StreamReader	Used for reading characters from a byte stream.
StreamWriter	Is used for writing characters to a stream.
StringReader	Is used for reading from a string buffer.
StringWriter	Is used for writing into a string buffer.

The FileStream Class:

The **FileStream** class in the System.IO namespace helps in reading from, writing to and closing files. This class derives from the abstract class Stream.

Syntax to create the object of filestream class

FileStream <object_name> = new FileStream(<file_name>, <FileMode Enumerator>, <FileAccess Enumerator>, <FileShare Enumerator>);



Parameter	Description
FileMode	The **FileMode** enumerator defines various methods for opening files. The members of the FileMode enumerator are: ▪ **Append**: It opens an existing file and puts cursor at the end of file, or creates the file, if the file does not exist. ▪ **Create**: It creates a new file. ▪ **CreateNew**: It specifies to the operating system, that it should create a new file. ▪ **Open**: It opens an existing file. ▪ **OpenOrCreate**: It specifies to the operating system that it should open a file if it exists, otherwise it should create a new file. ▪ **Truncate**: It opens an existing file and truncates its size to zero bytes.
FileAccess	**FileAccess** enumerators have members: **Read**, **ReadWrite** and **Write**.
FileShare	**FileShare** enumerators have the following members: ▪ **Inheritable**: It allows a file handle to pass inheritance to the child processes ▪ **None**: It declines sharing of the current file ▪ **Read**: It allows opening the file for reading ▪ **ReadWrite**: It allows opening the file for reading and writing ▪ **Write**: It allows opening the file for writing

StreamReader & StreamWriter Class:

The **StreamReader** and **StreamWriter** classes are used for reading from and writing data to text files. These classes inherit from the abstract base class Stream, which supports reading and writing bytes into a file stream.

The **StreamReader Class:**

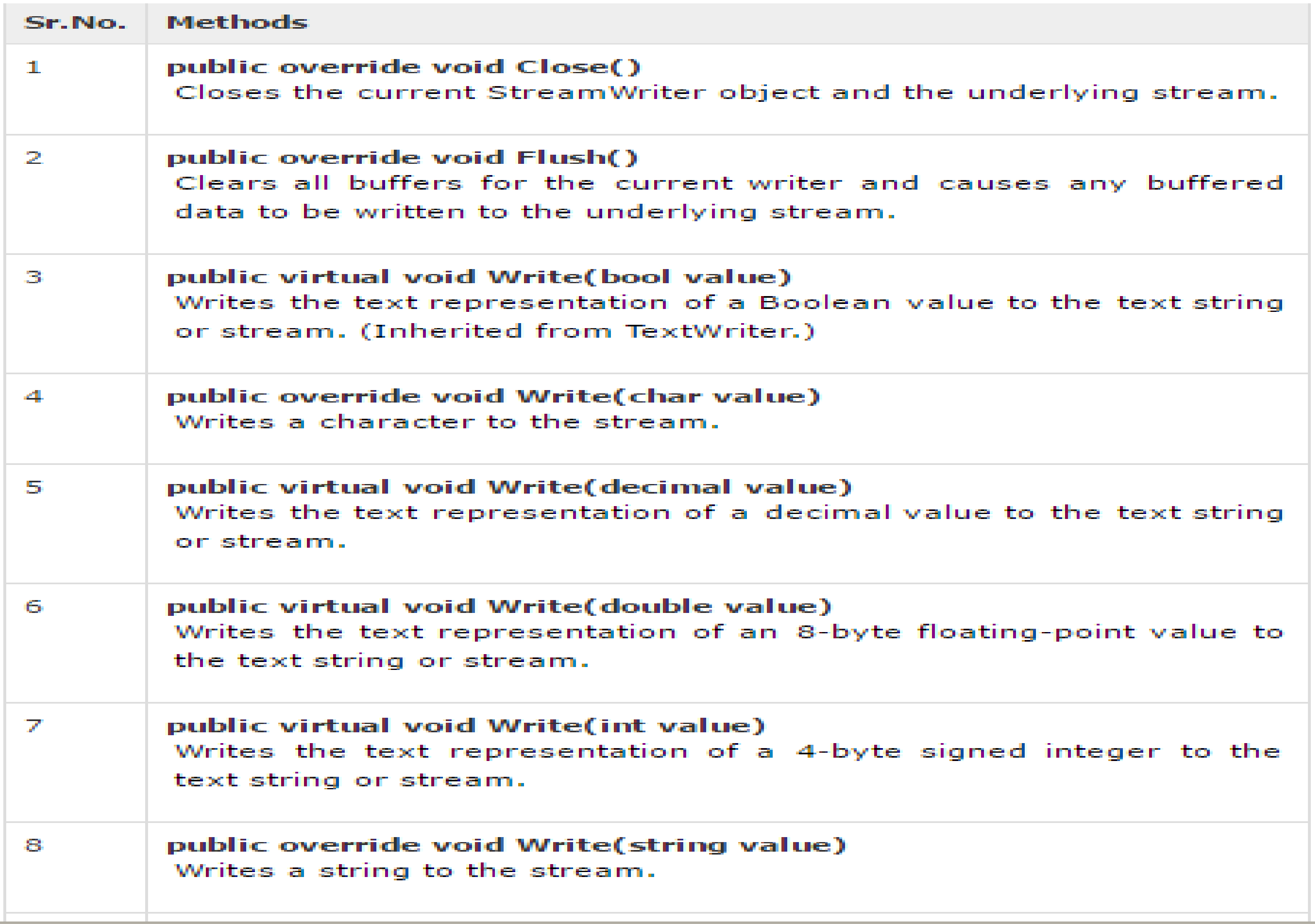
The StreamReader Class

The **StreamReader** class also inherits from the abstract base class TextReader that represents a reader for reading series of characters. The following table describes some of the commonly used **methods** of the StreamReader class:

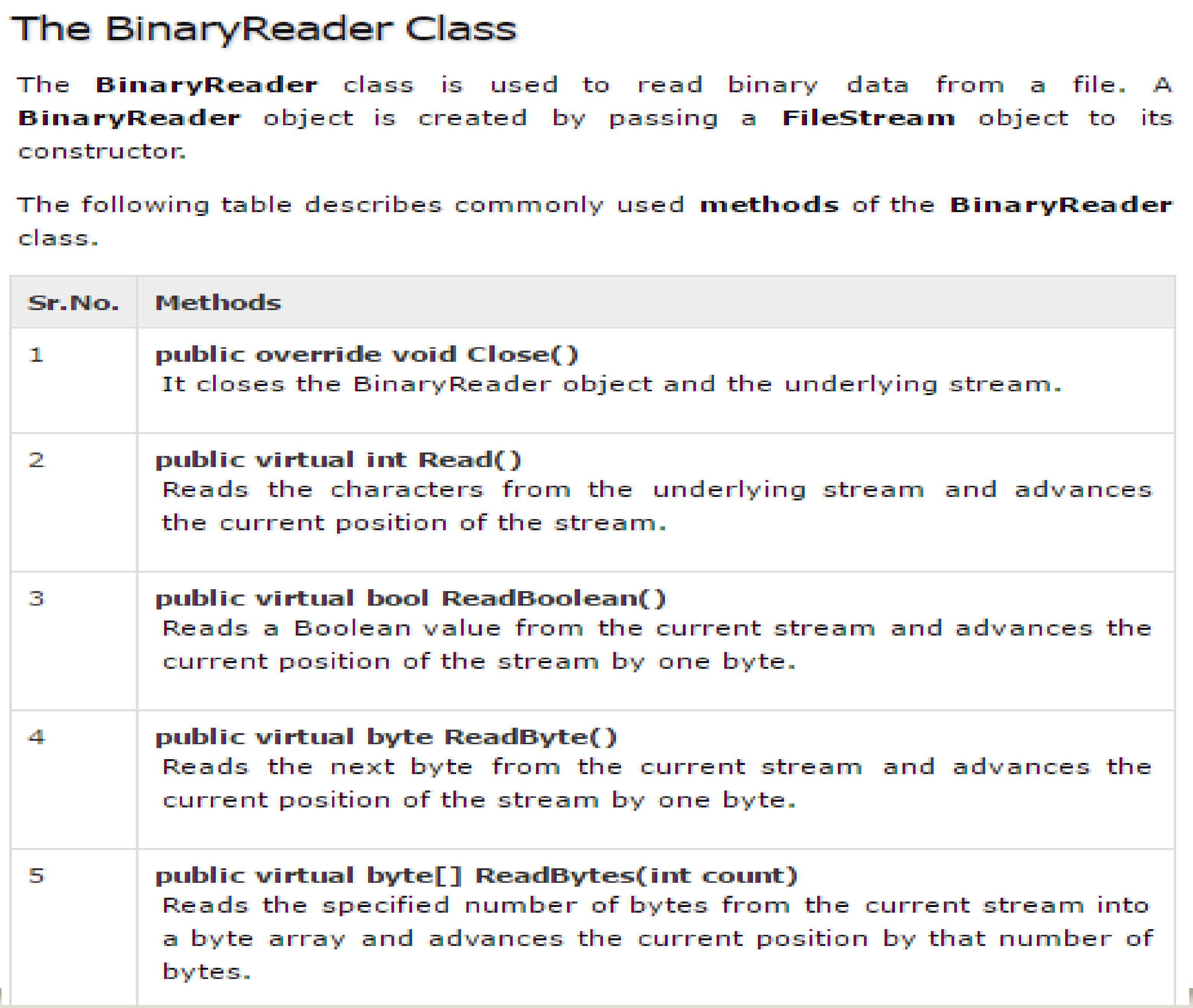
Sr.No.	Methods
1	**public override void Close()** It closes the StreamReader object and the underlying stream, and releases any system resources associated with the reader.
2	**public override int Peek()** Returns the next available character but does not consume it.
3	**public override int Read()** Reads the next character from the input stream and advances the character position by one.

The StreamWriter Class:

The **StreamWriter** class inherits from the abstract class TextWriter that represents a writer, which can write a series of character.



Sr.No.	Methods
1	**public override void Close()** Closes the current StreamWriter object and the underlying stream.
2	**public override void Flush()** Clears all buffers for the current writer and causes any buffered data to be written to the underlying stream.
3	**public virtual void Write(bool value)** Writes the text representation of a Boolean value to the text string or stream. (Inherited from TextWriter.)
4	**public override void Write(char value)** Writes a character to the stream.
5	**public virtual void Write(decimal value)** Writes the text representation of a decimal value to the text string or stream.
6	**public virtual void Write(double value)** Writes the text representation of an 8-byte floating-point value to the text string or stream.
7	**public virtual void Write(int value)** Writes the text representation of a 4-byte signed integer to the text string or stream.
8	**public override void Write(string value)** Writes a string to the stream.

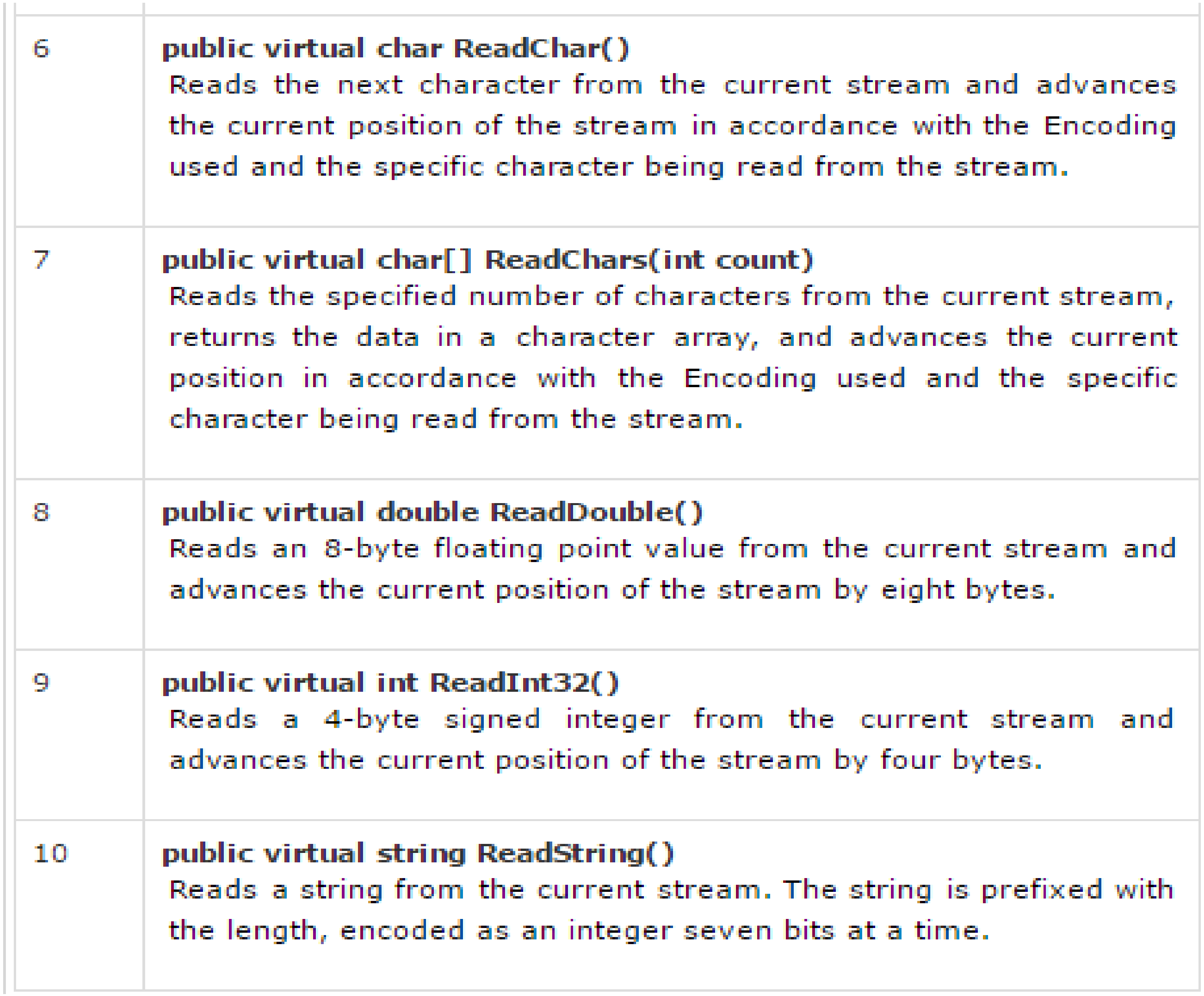


The BinaryReader Class

The **BinaryReader** class is used to read binary data from a file. A **BinaryReader** object is created by passing a **FileStream** object to its constructor.

The following table describes commonly used **methods** of the **BinaryReader** class.

Sr.No.	Methods
1	**public override void Close()** It closes the BinaryReader object and the underlying stream.
2	**public virtual int Read()** Reads the characters from the underlying stream and advances the current position of the stream.
3	**public virtual bool ReadBoolean()** Reads a Boolean value from the current stream and advances the current position of the stream by one byte.
4	**public virtual byte ReadByte()** Reads the next byte from the current stream and advances the current position of the stream by one byte.
5	**public virtual byte[] ReadBytes(int count)** Reads the specified number of bytes from the current stream into a byte array and advances the current position by that number of bytes.



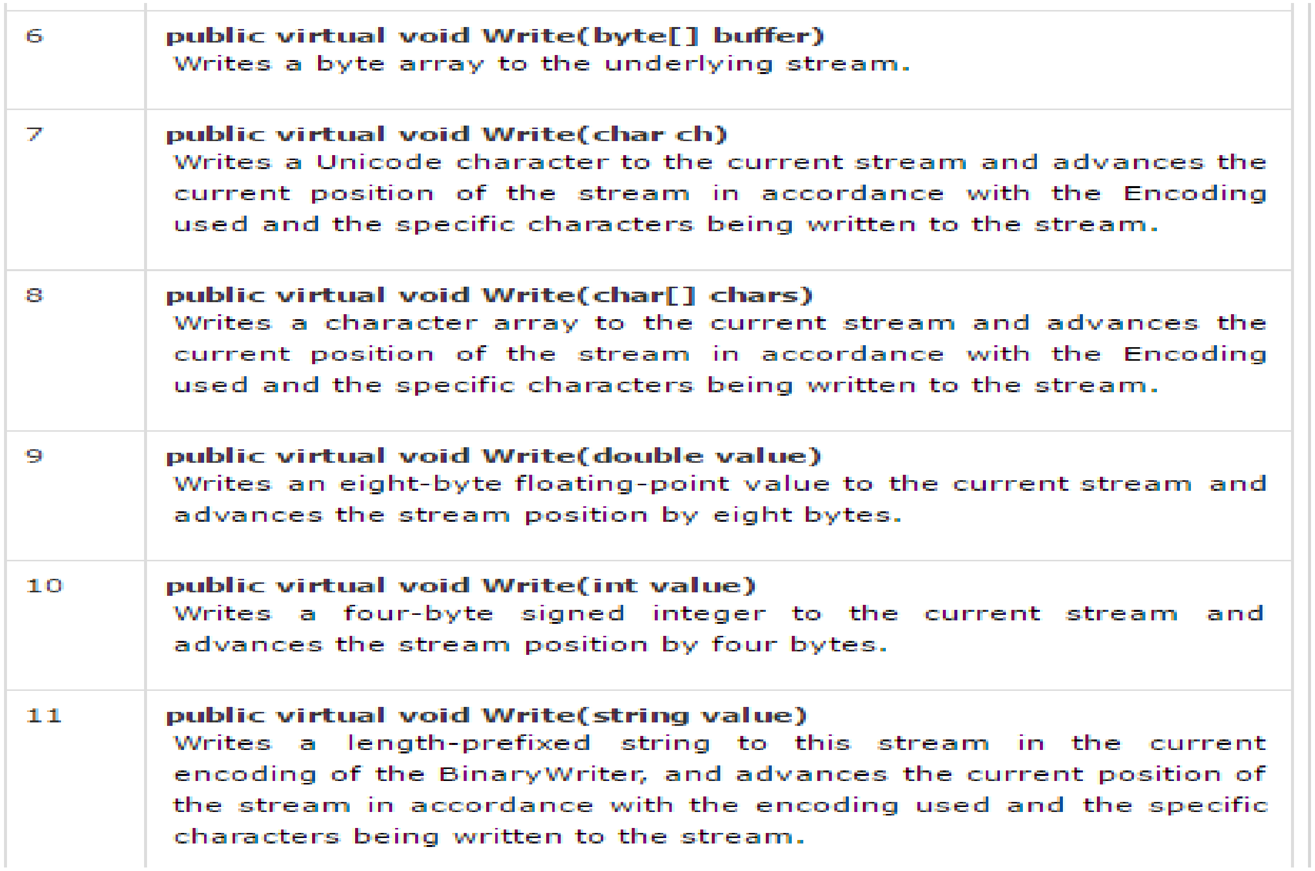
6	**public virtual char ReadChar()** Reads the next character from the current stream and advances the current position of the stream in accordance with the Encoding used and the specific character being read from the stream.
7	**public virtual char[] ReadChars(int count)** Reads the specified number of characters from the current stream, returns the data in a character array, and advances the current position in accordance with the Encoding used and the specific character being read from the stream.
8	**public virtual double ReadDouble()** Reads an 8-byte floating point value from the current stream and advances the current position of the stream by eight bytes.
9	**public virtual int ReadInt32()** Reads a 4-byte signed integer from the current stream and advances the current position of the stream by four bytes.
10	**public virtual string ReadString()** Reads a string from the current stream. The string is prefixed with the length, encoded as an integer seven bits at a time.

The BinaryWriter Class

The **BinaryWriter** class is used to write binary data to a stream. A BinaryWriter object is created by passing a FileStream object to its constructor.

The following table describes commonly used methods of the BinaryWriter class.

Sr.No.	Functions
1	**public override void Close()** It closes the BinaryWriter object and the underlying stream.
2	**public virtual void Flush()** Clears all buffers for the current writer and causes any buffered data to be written to the underlying device.
3	**public virtual long Seek(int offset, SeekOrigin origin)** Sets the position within the current stream.
4	**public virtual void Write(bool value)** Writes a one-byte Boolean value to the current stream, with 0 representing false and 1 representing true.
5	**public virtual void Write(byte value)** Writes an unsigned byte to the current stream and advances the stream position by one byte.



6	**public virtual void Write(byte[] buffer)** Writes a byte array to the underlying stream.
7	**public virtual void Write(char ch)** Writes a Unicode character to the current stream and advances the current position of the stream in accordance with the Encoding used and the specific characters being written to the stream.
8	**public virtual void Write(char[] chars)** Writes a character array to the current stream and advances the current position of the stream in accordance with the Encoding used and the specific characters being written to the stream.
9	**public virtual void Write(double value)** Writes an eight-byte floating-point value to the current stream and advances the stream position by eight bytes.
10	**public virtual void Write(int value)** Writes a four-byte signed integer to the current stream and advances the stream position by four bytes.
11	**public virtual void Write(string value)** Writes a length-prefixed string to this stream in the current encoding of the BinaryWriter, and advances the current position of the stream in accordance with the encoding used and the specific characters being written to the stream.

Advantages

Allow to read and write primitive types easily

Drawbacks

Writing/Reading objects needs to implement specific code

C# allows you to work with the directories and files using various directory and file related classes such as the **DirectoryInfo** class and the **FileInfo** class.

The DirectoryInfo Class

The **DirectoryInfo** class is derived from the **FileSystemInfo** class. It has various methods for creating, moving, and browsing through directories and subdirectories. This class cannot be inherited.

Following are some commonly used **properties** of the **DirectoryInfo** class:

Sr.No.	Properties
1	**Attributes** Gets the attributes for the current file or directory.
2	**CreationTime** Gets the creation time of the current file or directory.
3	**Exists** Gets a Boolean value indicating whether the directory exists.
4	**Extension** Gets the string representing the file extension.
5	**FullName** Gets the full path of the directory or file.

6	**LastAccessTime** Gets the time the current file or directory was last accessed.
7	**Name** Gets the name of this DirectoryInfo instance.

Following are some commonly used **methods** of the **DirectoryInfo** class:

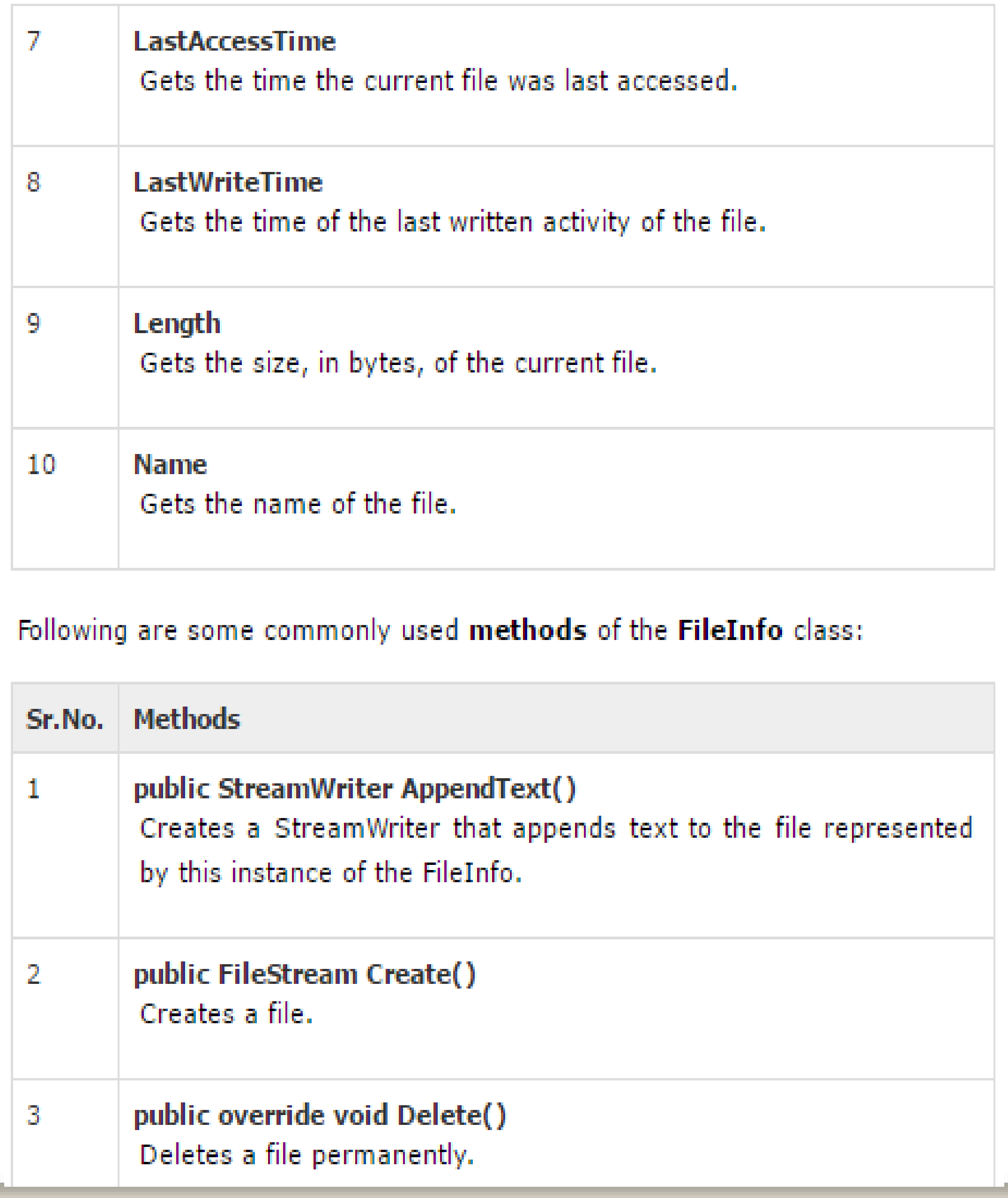
Sr.No.	Methods
1	**public void Create()** Creates a directory.
2	**public DirectoryInfo CreateSubdirectory(string path)** Creates a subdirectory or subdirectories on the specified path. The specified path can be relative to this instance of the DirectoryInfo class.
3	**public override void Delete()** Deletes this DirectoryInfo if it is empty.
4	**public DirectoryInfo[] GetDirectories()** Returns the subdirectories of the current directory.
5	**public FileInfo[] GetFiles()** Returns a file list from the current directory.

The FileInfo Class

The **FileInfo** class is derived from the **FileSystemInfo** class. It has properties and instance methods for creating, copying, deleting, moving, and opening of files, and helps in the creation of FileStream objects. This class cannot be inherited.

Following are some commonly used **properties** of the **FileInfo** class:

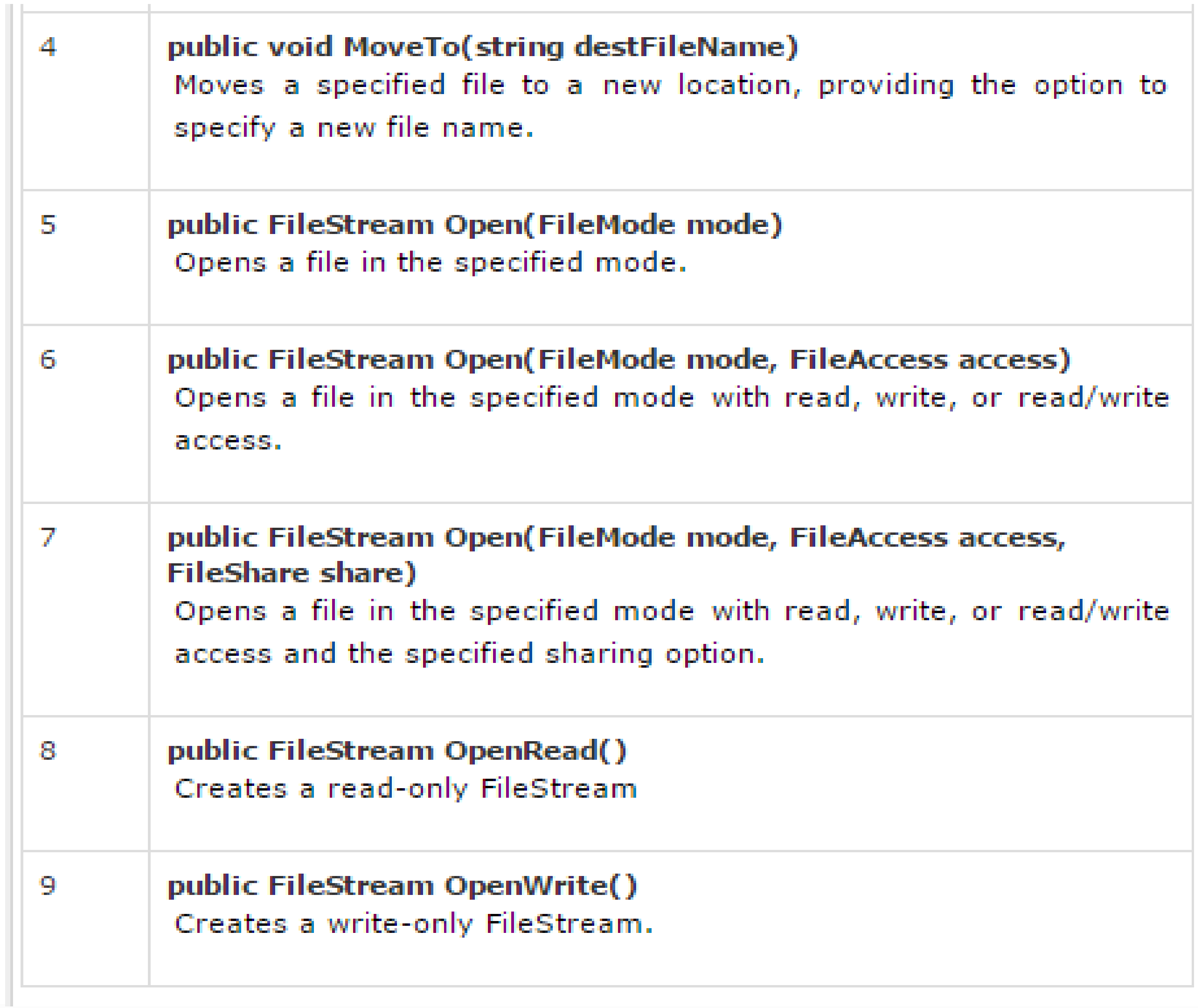
Sr.No.	Properties
1	**Attributes** Gets the attributes for the current file.
2	**CreationTime** Gets the creation time of the current file.
3	**Directory** Gets an instance of the directory which the file belongs to.
4	**Exists** Gets a Boolean value indicating whether the file exists.
5	**Extension** Gets the string representing the file extension.
6	**FullName** Gets the full path of the file.



7	**LastAccessTime** Gets the time the current file was last accessed.
8	**LastWriteTime** Gets the time of the last written activity of the file.
9	**Length** Gets the size, in bytes, of the current file.
10	**Name** Gets the name of the file.

Following are some commonly used **methods** of the **FileInfo** class:

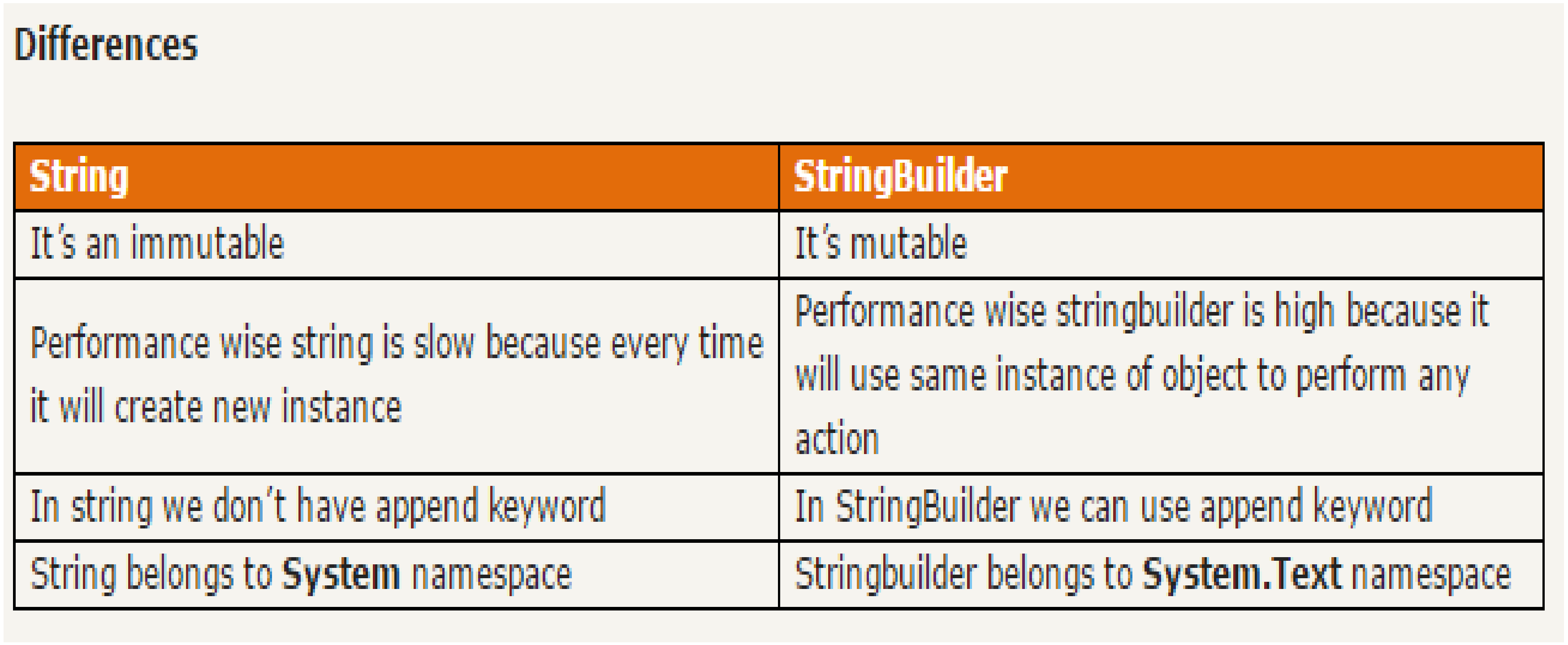
Sr.No.	Methods
1	**public StreamWriter AppendText()** Creates a StreamWriter that appends text to the file represented by this instance of the FileInfo.
2	**public FileStream Create()** Creates a file.
3	**public override void Delete()** Deletes a file permanently.



4	**public void MoveTo(string destFileName)** Moves a specified file to a new location, providing the option to specify a new file name.
5	**public FileStream Open(FileMode mode)** Opens a file in the specified mode.
6	**public FileStream Open(FileMode mode, FileAccess access)** Opens a file in the specified mode with read, write, or read/write access.
7	**public FileStream Open(FileMode mode, FileAccess access, FileShare share)** Opens a file in the specified mode with read, write, or read/write access and the specified sharing option.
8	**public FileStream OpenRead()** Creates a read-only FileStream
9	**public FileStream OpenWrite()** Creates a write-only FileStream.

String & Stringbuilder Class:

No. You can't. When initializing a StringBuilder, you are going down in performance. Also many actions that you do with string can't be done with StringBinder. Actually it is used mostly for situations as explained above. Last week, I showed a person who used StringBuilder to just add two strings together! It's really nonsense. We must really think about the overhead of initialization. In my personal experience, aStringBuilder can be used where more than four or more string concatenations take place. Also if you try to do some other manipulation (like removing a part from the string, replacing a part in the string, etc.), then it's better not to use StringBuilder at those places. This is because we are anyway creating new strings. Another important issue. We must be careful to guess the size of StringBuilder. If the size which we are going to get is more than what is assigned, it must increase the size. This will reduce its performance.



Differences

String	StringBuilder
It's an immutable	It's mutable
Performance wise string is slow because every time it will create new instance	Performance wise stringbuilder is high because it will use same instance of object to perform any action
In string we don't have append keyword	In StringBuilder we can use append keyword
String belongs to **System** namespace	Stringbuilder belongs to **System.Text** namespace

Thank you