#Enumeration and Enum type-

-Set of named set of constants

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

-Enum Comes under value type category.

[<modifiers>] enum <Name> [: <Type>]

{

-List of named constant

}

#Indexers-

1. This is like a property
2. This is declared inside the class
3. If you declare it inside the class then it make your class as virtual array.
4. You can access the fields in an array format.

[<modifiers>] <type> this[<parameter list>]

{

[get{<stmts>}] //Get Accessor

[set{<stmts>}] //Set Accessor

}

Pass By Value And Pass By Reference

In c#, Passing a [Value-Type](https://www.tutlane.com/tutorial/csharp/csharp-value-type-and-reference-type-with-examples" \o "C# Value Types with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank) parameter to a method by value means passing a copy of the variable to the method. So the changes made to the parameter inside of the called method will not have an effect on the original data stored in the argument variable.

As discussed earlier, [Value-Type](https://www.tutlane.com/tutorial/csharp/csharp-value-type-and-reference-type-with-examples" \o "C# Value Types with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank) variables will contain the value directly on its memory and [Reference-Type](https://www.tutlane.com/tutorial/csharp/csharp-value-type-and-reference-type-with-examples" \o "C# Reference Types with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank) variables will contain a reference of its data.

## C# Passing Parameters By Value Example

Following is the example of passing a [value type](https://www.tutlane.com/tutorial/csharp/csharp-value-type-and-reference-type-with-examples" \o "C# Value Types with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank) parameter to a method by value in c# programming language.

using System;

namespace Tutlane

{

    class Program

    {

        static void Main(string[] args)

        {

            int x = 10;

            Console.WriteLine("Variable Value Before Calling the Method: {0}", x);

            Multiplication(x);

            Console.WriteLine("Variable Value After Calling the Method: {0}", x);

            Console.WriteLine("Press Enter Key to Exit..");

            Console.ReadLine();

        }

        public static void Multiplication(int a)

        {

            a \*= a;

            Console.WriteLine("Variable Value Inside the Method: {0}", a);

        }

    }

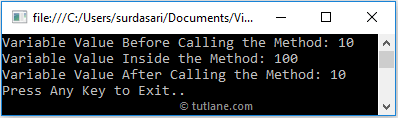
}

If you observe the above example, the variable ****x**** is a [value type](https://www.tutlane.com/tutorial/csharp/csharp-value-type-and-reference-type-with-examples" \o "C# Value Types with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank) and it passed to the ****Multiplication**** method. The content of variable ****x**** copied to the parameter ****a**** and made required modifications in the ****Multiplication**** method but the changes made inside of the method have no effect on the original value of the variable.

## Output of C# Passing Parameters By Value Example

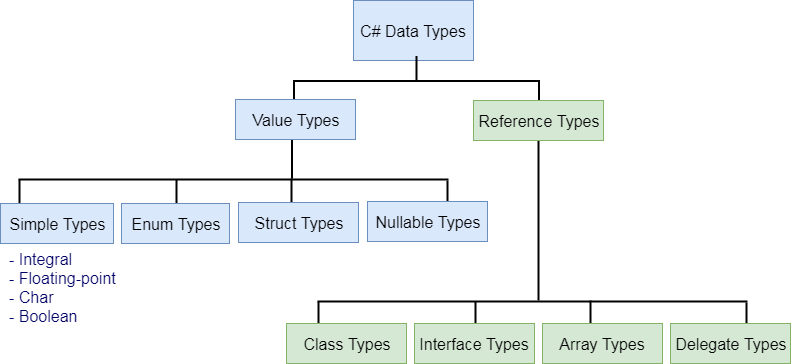
When we execute the above c# program, we will get the result as shown below.

 If you observe the above result, the variable value not changed even after we made the modifications in our method.



This is how we can pass parameters to the method by value in c# programming language based on our requirements.

Data Types:-

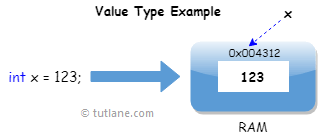


Value Type And Reference Type:-

1. In c#, we have two ways to allocate the space in memory, i.e. either on ****stack**** or ****heap**** memory based on the ****Value Type**** or ****Reference Type**** parameter
2. Value Type:-

These comes in value type

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| int | float | long | char | bool |
| byte | decimal | double | enum | sbyte |
| short | struct | uint | ulong | ushort |



For example, if we define and assign a value to the variable like int x = 123; then the system will use the same memory space of variable ‘****x****’ to store the value ‘****123****’.

Value Type and Reference Type

We have learned about the data types in the previous section. In C#, these data types are categorized based on how they store their value in the memory. C# includes following categories of data types:

1. Value type
2. Reference type
3. Pointer type

Here, we will learn about value types and reference types.

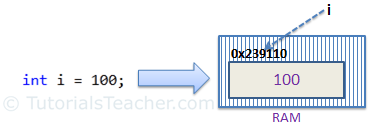
Value Type:

A data type is a value type if it holds a data value within its own memory space. It means variables of these data types directly contain their values.

IMG_261All the value types derive from *System.ValueType*, which in-turn, derives from *System.Object*.

For example, consider integer variable int i = 100;

The system stores 100 in the memory space allocated for the variable 'i'. The following image illustrates how 100 is stored at some hypothetical location in the memory (0x239110) for 'i':

[](./Value Type and Reference Type_files/value-type-memory-allocation.png" \t "_blank)Memory allocation for Value Type

The following data types are all of value type:

* bool
* byte
* char
* decimal
* double
* enum
* float
* int
* long
* sbyte
* short
* struct
* uint
* ulong
* ushort

Passing by Value:

When you pass a value type variable from one method to another method, the system creates a separate copy of a variable in another method, so that if value got changed in the one method won't affect on the variable in another method.

Example: Value Type

static void ChangeValue(int x)

{

x = 200;

Console.WriteLine(x);

}

static void Main(string[] args)

{

int i = 100;

Console.WriteLine(i);

ChangeValue(i);

Console.WriteLine(i);

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-ba76t1" \o "Try this example code yourself" \t "_blank)

Output:

100   
200   
100

In the above example, variable i in Main() method remains unchanged even after we pass it to the ChangeValue() method and change it's value there.

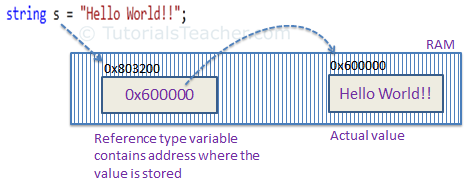
Reference Type

Unlike value types, a reference type doesn't store its value directly. Instead, it stores the address where the value is being stored. In other words, a reference type contains a pointer to another memory location that holds the data.

For example, consider following string variable:

string s = "Hello World!!";

The following image shows how the system allocates the memory for the above string variable.

[](./Value Type and Reference Type_files/raference-type-memory-allocation.png" \t "_blank)Memory allocation for Reference type

As you can see in the above image, the system selects a random location in memory (0x803200) for the variable 's'. The value of a variable s is 0x600000 which is the memory address of the actual data value. Thus, reference type stores the address of the location where the actual value is stored instead of value itself.

The following data types are of reference type:

* String
* All arrays, even if their elements are value types
* Class
* Delegates

Pass by Reference

When you pass a reference type variable from one method to another, it doesn't create a new copy; instead, it passes the address of the variable. If we now change the value of the variable in a method, it will also be reflected in the calling method.

Example: Reference Type Variable

static void ChangeReferenceType(Student std2)

{

std2.StudentName = "Steve";

}

static void Main(string[] args)

{

Student std1 = new Student();

std1.StudentName = "Bill";

ChangeReferenceType(std1);

Console.WriteLine(std1.StudentName);

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-WL8Xgk" \o "Try this example code yourself" \t "_blank)

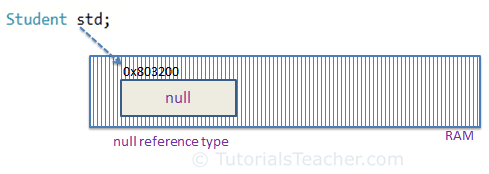
Output:

Steve

In the above example, since Student is an object, when we send the Student object std1 to the ChangeReferenceType() method, what is actually sent is the memory address of std1. Thus, when the ChangeReferenceType() method changes StudentName, it is actually changing StudentName of std1, because std1 and std2 are both pointing to the same address in memory. Therefore, the output is Steve.

Null

Reference types have null value by default, when they are not initialized. For example, a string variable (or any other variable of reference type datatype) without a value assigned to it. In this case, it has a null value, meaning it doesn't point to any other memory location, because it has no value yet.

[](./Value Type and Reference Type_files/null.png" \t "_blank)Null Reference type

A value type variable cannot be null because it holds a value not a memory address. However, value type variables must be assigned some value before use. The compiler will give an error if you try to use a local value type variable without assigning a value to it.

Example: Compile Time Error

void someFunction()

{

int i;

Console.WriteLine(i);

}

IMG_261C# 2.0 introduced nullable types for value types so that you can assign null to a value type variable or declare a value type variable without assigning a value to it.

However, value type field in a class can be declared without initialization (field not a local variable in the function) . It will have a default value if not assigned any value, e.g., int will have 0, boolean will have false and so on.

Example: Value Type Field

class myClass

{

public int i;

}

myClass mcls = new myClass();

Console.WriteLine(mcls.i);

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-S9ne6B" \o "Try this example code yourself" \t "_blank)

Output:

0

IMG_262Points to Remember :

1. Value type stores the value in its memory space, whereas reference type stores the address of the value where it is stored.
2. Primitive data types and struct are of the 'Value' type. Class objects, string, array, delegates are reference types.
3. Value type passes byval by default. Reference type passes byref by default.
4. Value types and reference types stored in Stack and Heap in the memory depends on the scope of the variable.

Further Reading

* Read Eric Lippert's blog [part 1](https://blogs.msdn.com/b/ericlippert/archive/2009/04/27/the-stack-is-an-implementation-detail.aspx" \t "_blank) and [part 2](https://blogs.msdn.com/b/ericlippert/archive/2009/05/04/the-stack-is-an-implementation-detail-part-two.aspx" \t "_blank) for more details about value type and reference type.
* Visit MSDN to know [default values of value types](https://msdn.microsoft.com/en-us/library/83fhsxwc.aspx" \o "Default Values of Value Types" \t "_blank).

# Working with Files & Directories in C#

C# provides the following classes to work with the File system. They can be used to access directories, access files, open files for reading or writing, create a new file or move existing files from one location to another, etc.

| Class Name | Usage |
| --- | --- |
| [File](https://www.tutorialsteacher.com/csharp/csharp-file" \l "file) | File is a static class that provides different functionalities like copy, create, move, delete, open for reading or /writing, encrypt or decrypt, check if a file exists, append lines or text to a file’s content, get last access time, etc. |
| [FileInfo](https://www.tutorialsteacher.com/csharp/csharp-fileinfo) | The FileInfo class provides the same functionality as a static File class. You have more control on how you do read/write operations on a file by writing code manually for reading or writing bytes from a file. |
| [Directory](https://msdn.microsoft.com/en-us/library/system.io.directory(v=vs.110).aspx" \t "https://www.tutorialsteacher.com/csharp/_blank) | Directory is a static class that provides functionality for creating, moving, deleting and accessing subdirectories. |
| [DirectoryInfo](https://msdn.microsoft.com/en-us/library/system.io.directoryinfo(v=vs.110).aspx" \t "https://www.tutorialsteacher.com/csharp/_blank) | DirectoryInfo provides instance methods for creating, moving, deleting and accessing subdirectories. |
| [Path](https://msdn.microsoft.com/en-us/library/system.io.path(v=vs.110).aspx" \t "https://www.tutorialsteacher.com/csharp/_blank) | Path is a static class that provides functionality such as retrieving the extension of a file, changing the extension of a file, retrieving the absolute physical path, and other path related functionalities. |

## File

C# includes static **File** class to perform I/O operation on physical file system. The static File class includes various utility method to interact with physical file of any type e.g. binary, text etc.

Use this static File class to perform some quick operation on physical file. It is not recommended to use File class for multiple operations on multiple files at the same time due to performance reasons. Use FileInfo class in that scenario.

## Important Methods of Static File Class

| Method | Usage |
| --- | --- |
| AppendAllLines | Appends lines to a file, and then closes the file. If the specified file does not exist, this method creates a file, writes the specified lines to the file, and then closes the file. |
| AppendAllText | Opens a file, appends the specified string to the file, and then closes the file. If the file does not exist, this method creates a file, writes the specified string to the file, then closes the file. |
| AppendText | Creates a StreamWriter that appends UTF-8 encoded text to an existing file, or to a new file if the specified file does not exist. |
| Copy | Copies an existing file to a new file. Overwriting a file of the same name is not allowed. |
| Create | Creates or overwrites a file in the specified path. |
| CreateText | Creates or opens a file for writing UTF-8 encoded text. |
| Decrypt | Decrypts a file that was encrypted by the current account using the Encrypt method. |
| Delete | Deletes the specified file. |
| Encrypt | Encrypts a file so that only the account used to encrypt the file can decrypt it. |
| Exists | Determines whether the specified file exists. |
| GetAccessControl | Gets a FileSecurity object that encapsulates the access control list (ACL) entries for a specified file. |
| Move | Moves a specified file to a new location, providing the option to specify a new file name. |
| Open | Opens a FileStream on the specified path with read/write access. |
| ReadAllBytes | Opens a binary file, reads the contents of the file into a byte array, and then closes the file. |
| ReadAllLines | Opens a text file, reads all lines of the file, and then closes the file. |
| ReadAllText | Opens a text file, reads all lines of the file, and then closes the file. |
| Replace | Replaces the contents of a specified file with the contents of another file, deleting the original file, and creating a backup of the replaced file. |
| WriteAllBytes | Creates a new file, writes the specified byte array to the file, and then closes the file. If the target file already exists, it is overwritten. |
| WriteAllLines | Creates a new file, writes a collection of strings to the file, and then closes the file. |
| WriteAllText | Creates a new file, writes the specified string to the file, and then closes the file. If the target file already exists, it is overwritten. |

## Append Text Lines

Use AppendAllLines() method to append multiple text lines to the specified file as shown below.

Example: Append all text lines to a file

string dummyLines = "This is first line." + Environment.NewLine +

"This is second line." + Environment.NewLine +

"This is third line.";

//Opens DummyFile.txt and append lines. If file is not exists then create and open.File.AppendAllLines(@"C:\DummyFile.txt", dummyLines.Split(Environment.NewLine.ToCharArray()).ToList<string>());

## Append String

Use *File.AppendAllText()* method to append string to a file in single line of code as shown below.

Example: Append string to a file

//Opens DummyFile.txt and append Text. If file is not exists then create and open.File.AppendAllText(@"C:\ DummyFile.txt", "This is File testing");

## Overwrite Text

Use *File.WriteAllText()* method to write texts to the file. Please note that it will not append text but overwrite existing texts.

Example: Overwrite existing texts

//Opens DummyFile.txt and write texts. If file is not exists then create and open.File.WriteAllText(@"C:\DummyFile.txt", "This is dummy text");

The following example shows how to perform different operations using static File class.

Example: Multiple File operations

//Check whether file is exists or not at particular locationbool isFileExists = File.Exists(@"C:\ DummyFile.txt"); // returns false

//Copy DummyFile.txt as new file DummyFileNew.txtFile.Copy(@"C:\DummyFile.txt", @"D:\NewDummyFile.txt");

//Get when the file was accessed last time DateTime lastAccessTime = File.GetLastAccessTime(@"C:\DummyFile.txt");

//get when the file was written last timeDateTime lastWriteTime = File.GetLastWriteTime(@"C:\DummyFile.txt");

// Move file to new locationFile.Move(@"C:\DummyFile.txt", @"D:\DummyFile.txt");

//Open file and returns FileStream for reading bytes from the fileFileStream fs = File.Open(@"D:\DummyFile.txt", FileMode.OpenOrCreate);

//Open file and return StreamReader for reading string from the fileStreamReader sr = File.OpenText(@"D:\DummyFile.txt");

//Delete fileFile.Delete(@"C:\DummyFile.txt");

Thus, it is easy to work with physical file using static File class. However, if you want more flexibility then use FileInfo class. The same way, use static Directory class to work with physical directories

1)Encapsulation-

# #Properties Vs Fields In C#

## ****Fields****are normal variable members of a class. ****Properties****are an abstraction to get and set their values. In this quick tutorial, you will understand the difference between them, and which one to use.

# **Fields**

*Fields*are normal variable members of a class. Generally, you should declare your fields as private, then use *Properties*to get and set their values. By this way you won’t affect their values them directly. This is common case practice since having public members violates the Encapsulation concept in OOP.

public class Student  
 {  
 private int \_id;  
 private string \_name;  
 }

# **Properties**

They are actually special methods called “*accessors*”. *Properties*are called *accessors* because they offer a way to get and set a field if you have a private field. They have two codes inside; *set{};* and*get{};* called “*[property accessors](https://msdn.microsoft.com/en-us/library/aa287786(v=vs.71).aspx" \t "https://medium.com/omarelgabrys-blog/_blank)*”.

public class Student

{

private int \_id;

private string \_name;

public int Id

{

get

{

return \_id;

}

set

{

\_id = value;

}

}

public string Name

{ get

{

return \_name;

}

set

{

\_name = value;

}

}

}*“value*” is a keyword, It refers to the assigned value, It’s like a parameter for the set method, The final code snippet will make it more obvious.

*Properties* can be used to read only or write only other fields. This could be done by declaring only either *get{}* or *set{}*. Also they can have access modifiers, like *private*, so you can only get or set their values inside their class.

*In*Java*, We need to declare*getters*and*setters*methods, But, In*C#*, You can defined a property for each field.*

#Static and non static:-

>static class can have only static members but non-static may have static and non-static members

Example-

static classes- Currency converter,mathematical operations ,conversions,console class,and math class

Non static classes- Student, accounts

# **#Constructors:-**

In c#, ****Constructor**** is a method which will invoke automatically whenever an instance of [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples" \o "C# Classes and Objects with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank) or ****struct**** is created.  The constructor will have the same name as the [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples" \o "C# Classes and Objects with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank) or ****struct**** and it useful to initialize and set default values for the data members of the new object.

In case, if we create a class without having any constructor, then the compiler will automatically create a one default constructor for that class. So, there is always one constructor that will exist in every class.

In c#, a [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples" \o "C# Classes and Objects with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank) can contain more than one constructor with different types of arguments and the constructors will never return anything, so we don’t need to use any return type, not even ****void**** while defining the constructor method in the [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples" \o "C# Classes and Objects with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank).

## C# Constructor Syntax

As discussed, the constructor is a method and it won’t contain any return type. If you want to create a constructor in c#, then you need to create a method with the class name

Following is the syntax of creating a constructor in c# programming language.

public class User

{

// Constructor

public User()

{

// Your Custom Code

}

}

If you observe the above syntax, we created a class called “User” and a method whose name is same as the class name. Here the method User() will become a constructor of our class.

## C# Constructor Types

In c#, we have a different type of constructors available, those are

* [Default Constructor](https://www.tutlane.com/tutorial/csharp/csharp-constructors-with-examples" \l "divcsdfcstr" \o "C# Default Constructor with Examples)
* [Parameterized Constructor](https://www.tutlane.com/tutorial/csharp/csharp-constructors-with-examples" \l "divcspzcst" \o "C# Parameterized Constructor with Examples)
* [Copy Constructor](https://www.tutlane.com/tutorial/csharp/csharp-copy-constructor-with-examples" \o "C# Copy Constructor with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank)
* [Static Constructor](https://www.tutlane.com/tutorial/csharp/csharp-static-constructor-with-examples" \o "C# Static Constructor with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank)
* [Private Constructor](https://www.tutlane.com/tutorial/csharp/csharp-private-constructor-with-examples" \o "C# Private Constructor with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank)

1. **Default constructors->**

**Called automatically even if you don’t create it And you can set default values in default constructor**

1. **Parameter constructor-> It takes one or more parameters and we can create the multiple overload**

**Static constructor->**

## Static Constructor in C#

When a constructor is created using a static keyword, it will be invoked only once for all of the instances of the class and it is invoked during the creation of the first instance of the class or the first reference to a static member in the class. A static constructor is used to initialize static fields of the class and to write the code that needs to be executed only once.

**Some key points of a static constructor are:**

1. Static constructor called only one in his linespan.
2. A static constructor does not take access modifiers or have parameters.
3. A static constructor is called automatically to initialize the class before the first instance is created or any static members are referenced.
4. A static constructor cannot be called directly.
5. The user has no control over when the static constructor is executed in the program.
6. A typical use of static constructors is when the class is using a log file and the constructor is used to write entries to this file.
7. Static Constructor implicitly called
8. **Don’t take any parameter not allowed**

**Use OF Static Constructor-**

1. **Setting the static fields once eg:-Logging data to database**

1. **Private constructor->**To avoid Creation of Instances

->IF Class has static members only then we need private constructor

**->Class which having private constructor cannot be inherited**

**->we cannot create the object of a class which is having private constructor**

**->Use:-cannot be inherited**

**Isolated class**

**Application->Help or help library**

**#Inheritance And Constructor-**

1. **Whenever the child class instance is created then the base class constructor is called**

**first.**

1. **Base Class Constructors are not inherited**
2. **The field which is initialized in base class cant be set in derived class constructor so simply we use :base(parameter) which calls the base constructor and set the base class field.**

**ReadOnly:-**

In c#, ****readonly**** is a keyword which is useful to define read-only fields in our applications.  The read-only field values need to be initialized either at the declaration or in a constructor of the same class unlike [constant keyword in c#](https://www.tutlane.com/tutorial/csharp/csharp-const-constant-keyword" \o "C# Constant Keyword with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank). If we use readonly keyword with fields, then those field values will be evaluated at the runtime.

#Types of Exception-

|  |  |
| --- | --- |
| **Exception Class** | **Cause** |
| SystemException | A failed run-time check;used as a base class for other. |
| AccessException | Failure to access a type member, such as a method or field. |
| ArgumentException | An argument to a method was invalid. |
| ArgumentNullException | A null argument was passed to a method that doesn't accept it. |
| ArgumentOutOfRangeException | Argument value is out of range. |
| ArithmeticException | Arithmetic over - or underflow has occurred. |
| ArrayTypeMismatchException | Attempt to store the wrong type of object in an array. |
| BadImageFormatException | Image is in the wrong format. |
| CoreException | Base class for exceptions thrown by the runtime. |
| DivideByZeroException | An attempt was made to divide by zero. |
| FormatException | The format of an argument is wrong. |
| IndexOutOfRangeException | An array index is out of bounds. |
| InvalidCastExpression | An attempt was made to cast to an invalid class. |
| InvalidOperationException | A method was called at an invalid time. |
| MissingMemberException | An invalid version of a DLL was accessed. |
| NotFiniteNumberException | A number is not valid. |
| NotSupportedException | Indicates sthat a method is not implemented by a class. |
| NullReferenceException | Attempt to use an unassigned reference. |
| OutOfMemoryException | Not enough memory to continue execution. |
| StackOverflowException | A stack has overflown. |

#Access Modifiers-

1. We cannot use Private ,Protected , Protected internal on the class.
2. You can use only Internal and Public on the class
3. If You don’t Put anything before class then it is Internal Class
4. The member or methods in the class are private by default

Private- within the containing class only;

Internal- within the project both from child class or Instance

Protected-within the class and child class

Protected Internal- within the project or Child of other project

Public - Globally

**#Inheritance And Composition**

#Relationships

1. Is -a Relationship -Inheritance (University Teacher)-Weakly associated
2. Has -a Relationship-Composition (University Department)-Strongly associated

**#Method Overriding-**

In c#, ****Method Overriding**** means override a base class method in the derived [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples" \o "Classes and Objects in C# with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank) by creating a [method](https://www.tutlane.com/tutorial/csharp/csharp-methods-functions-with-examples" \o "Methods / Functions in C# with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank) with the same name and signatures to perform a different task. The Method Overriding in c# can be achieved by using ****override**** & ****virtual**** keywords along with the [inheritance](https://www.tutlane.com/tutorial/csharp/csharp-inheritance" \o "Inheritance in C# with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank) principle.

Suppose, if we want to change the behavior of the base class [method](https://www.tutlane.com/tutorial/csharp/csharp-methods-functions-with-examples" \o "Methods / Functions in C# with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank) in a derived [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples" \o "Classes and Objects in C# with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank), then we need to use ****method overriding****. The base class [method](https://www.tutlane.com/tutorial/csharp/csharp-methods-functions-with-examples" \o "Methods / Functions in C# with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank) which we want to override in the derived [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples" \o "Classes and Objects in C# with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank) that needs to be defined with ****virtual**** keyword and we need to use ****override**** keyword in derived [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples" \o "Classes and Objects in C# with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank) while defining the [method](https://www.tutlane.com/tutorial/csharp/csharp-methods-functions-with-examples" \o "Methods / Functions in C# with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank) with the same name and parameters then only we can override the base class method in a derived [class](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples" \o "Classes and Objects in C# with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank).

In c#, the Method Overriding is also called as ****run time polymorphism**** or ****late binding****. Following is the code snippet of implementing a ****method overriding**** in c# programming language.

// Base Class

public class Users

{

    public virtual void GetInfo()

    {

        Console.WriteLine("Base Class");

    }

}

// Derived Class

public class Details : Users

{

    public override void GetInfo()

    {

        Console.WriteLine("Derived Class");

    }

}

If you observe above code snippet, we created a two classes (“****Users****”, “****Details****”) and the derived class (Details) is [inheriting](https://www.tutlane.com/tutorial/csharp/csharp-inheritance" \o "Inheritance in C# with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank) the [properties](https://www.tutlane.com/tutorial/csharp/csharp-properties-get-set" \o "Properties in C# with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank) from base class (****Users****) and we are overriding the base class method ****GetInfo**** in derived class by creating a method with same name and parameters, this is called a ****method overriding**** in c#.

Here, we defined a ****GetInfo**** method with a ****virtual**** keyword in the base class to allow derived class to override that [method](https://www.tutlane.com/tutorial/csharp/csharp-methods-functions-with-examples" \o "Methods / Functions in C# with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank) using the ****override**** keyword.

As discussed, only the methods with a ****virtual**** keyword in the base class are allowed to override in derived class using ****override**** keyword.

3)Polymorphism

Generally, the polymorphism is a combination of two words, one is **poly** and another one is **morphs**. Here **poly** means “**multiple**” and **morphs** means “**forms**” so polymorphism means many forms.

In c#, polymorphism provides an ability for the [classes](https://www.tutlane.com/tutorial/csharp/csharp-classes-and-objects-with-examples" \o "Classes and Objects in C# with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank) to implement different [methods](https://www.tutlane.com/tutorial/csharp/csharp-methods-functions-with-examples" \o "Methods / Functions in C# with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank) that are called through the same name and it also provides an ability to invoke the [methods](https://www.tutlane.com/tutorial/csharp/csharp-methods-functions-with-examples" \o "Methods / Functions in C# with Examples" \t "https://www.tutlane.com/tutorial/csharp/_blank) of a derived class through base class reference during runtime based on our requirements.

In c#, we have two different kinds of polymorphisms available, those are

* Compile Time Polymorphism
* Run Time Polymorphism

## Compile Time Polymorphism Example-

Is nothing but method overloading and is achieve as follows:

public class Calculate

{

    public void AddNumbers(int a, int b)

    {

        Console.WriteLine("a + b = {0}", a + b);

    }

    public void AddNumbers(int a, int b, int c)

    {

        Console.WriteLine("a + b + c = {0}", a + b + c);

    }

}

1. Run Time Polymorphism-

Is achieved by using the method overriding-

public class Users

{

    public virtual void GetInfo()

    {

        Console.WriteLine("Base Class");

    }

}

// Derived Class

public class Details : Users

{

    public override void GetInfo()

    {

        Console.WriteLine("Derived Class");

    }

}

#Method Overloading And Method Overriding-

Overload-

1. We define multiple methods with the same name but by changing their parameters.
2. This can be performed within class as well as between parent and child class.
3. Overloading parent class methods in child class dose not require any permission from parent

4)It is all about defining multiple behaviors to a method

Overriding-

1. we define multiple methods with same name and same parameter
2. This can be performed only in child class
3. Overriding parent class methods in child class require permission from parent

Class (Which is virtual keyword on the method)

1. Overriding is changing the behavior of the parent method under the child method.

#Method Hiding-

1. Method hiding is also re-implementing the parent class method in the child class with same name and method signature .
2. I) In First case child class re-implement its parents method which is declared as Virtual

2)In second case child class re- implement its parent method which are not declared as virtual. (Using new keyword)

We Can Re-Implement Parent class method in child class using two methods:-

1. Method Overriding.
2. Method Hiding

#If you want to call parent class method along with the child class method there are two approaches

1. Creating the instance of parent .

### Or by creating the public class in child class and using base keyword to call parent method

#Abstract Class And Members.

-> A method without method body is know as abstract method , What method consist of is only the method declaration .

->IF method declared abstract under any class the child class of that class is responsible for implementing the method.

->If you declare a method as abstract then the class containing that method must be abstract.

Abstract class parent

{

Abstract method();

Non-abstract method();

}

Note: You Cannot Create a Instance of a abstract class.

->Abstract Class And Child class-

1. Implement each and every abstract method in parent class
2. Now only we can consume the non- abstract methods.

#Sealed Class And Sealed Members-

1. If you apply sealed modifier on class then that class can’t be inherited.
2. If you apply sealed modifier on method then you can’t override that method in it’s child class.

NOTE:-you can apply sealed only on methods which can be override.

#Interface-

Class: It’s a user-defined data type.

Interface:- This is also an user defined data type only.

Class : Contains Non-Abstract Methods(method with method body)

Abstract Class: Non-Abstract methods and also Abstract methods(method without method body)

Interface:-Contains only Abstract Method

Note: Every abstract method of an interface should be implemented by the child class of the interface without Fail maindatory

->Default scope of member in interface is public

->By Default the member are abstract in interface.

->We Can not declare fields in interface.

->If required an interface can inherit from another interface.

->You cannot crate an instance of a interface.

->But you can crate reference of interface.

#Types of inheritance supported by classes in c#

->single

->multilevel

->Hierarchical

#Hybrid and Multiple inheritance is not supported by classes.

#But Multiple Inheritance is supported by Interfaces

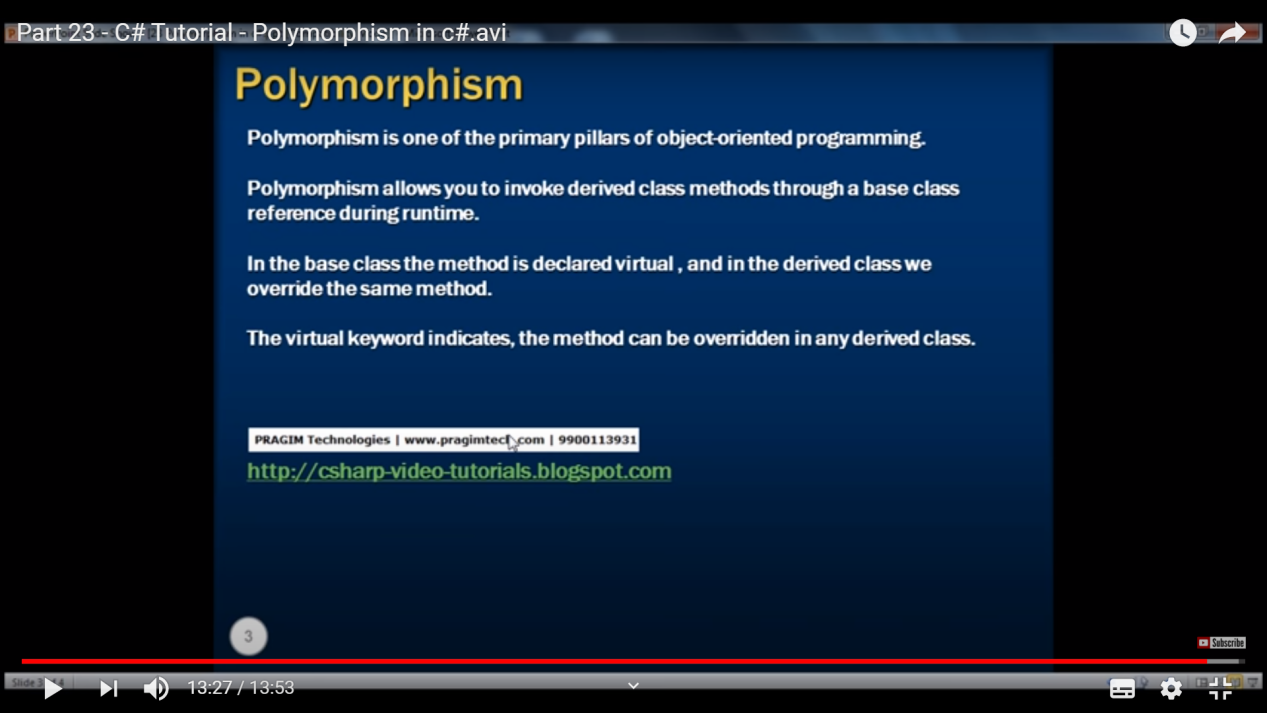
#A class can have one parent as well as set of interfaces.

#Upcasting And Downcasting-

--> Upcasting converts an object of a specialized type to a more general type

--> Downcasting converts an object from a general type to a more specialized type

#Polymorphism-



# Boxing and Unboxing in C#

[C#](https://www.tutorialsteacher.com/articles?category=csharp)

By TutorialsTeacher

 26 Sep 2019

C# has two kinds of data types, [value types and reference types](https://www.tutorialsteacher.com/csharp/csharp-value-type-and-reference-type" \t "https://www.tutorialsteacher.com/articles/_blank). Value type stores the value itself, whereas the reference type stores the address of the value where it is stored. Some predefined data types such as int, float, double, decimal, bool, char, etc. are value types and object, string, and array are reference types.

While working with these data types, you often need to convert value types to reference types or vice-versa. Because, both have different characteristics and .NET stores them differently in the memory, it must do some work internally to convert them from one type to another. These conversion processes are called boxing and unboxing.

## What is boxing?

Boxing is the process of converting a value type to the object type or any interface type implemented by this value type. Boxing is implicit.

Example: Boxing

int i = 10;object o = i; //performs boxing

In the above example, the integer variable i is assigned to object o. Since object type is a reference type and base class of all the classes in C#, an int can be assigned to an object type. This process of converting int to object is called boxing.

Let's look at a more practical example.

Example: Boxing

ArrayList list = new ArrayList();

list.Add(10); // boxing

list.Add("Bill");

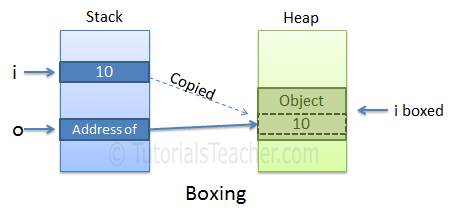
Above, ArrayList is a class in C#, and so it is a reference type. We add an int value 10 in it. So, .NET will perform the boxing process here to assign value type to reference type.

## Why named boxing?

You may be wondering, why is it named as boxing?

As you know, all the reference types stored on heap where it contains the address of the value and value type is just an actual value stored on the stack. Now, as shown in the first example, int i is assigned to object o. Object o must be an address and not a value itself. So, the CLR boxes the value type by creating a new System.Object on the heap and wraps the value of i in it and then assigns an address of that object to o. So, because the CLR creates a box on the heap that stores the value, the whole process is called 'Boxing'.

The following figure illustrates the boxing process.

[](https://www.tutorialsteacher.com/Content/images/articles/csharp/boxing.PNG)

## What is Unboxing?

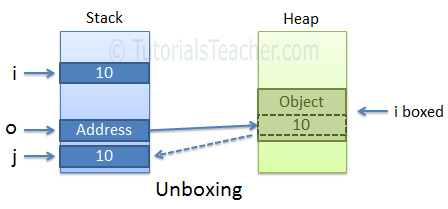
Unboxing is the reverse of boxing. It is the process of converting a reference type to value type. Unboxing extract the value from the reference type and assign it to a value type.

Unboxing is explicit. It means we have to cast explicitly.

Example: Unboxing

object o = 10;int i = (int)o; //performs unboxing

The following figure illustrates the unboxing process.

[](https://www.tutorialsteacher.com/Content/images/articles/csharp/unboxing.PNG)

A boxing conversion makes a copy of the value. So, changing the value of one variable will not impact others.

int i = 10;object o = i; // boxing

o = 20;

Console.WriteLine(i); // output: 10

The casting of a boxed value is not permitted. The following will throw an exception.

Example: Invalid Conversion

int i = 10;object o = i; // boxingdouble d = (double)o; // runtime exception

First do unboxing and then do casting, as shown below.

Example: Valid Conversion

int i = 10;object o = i; // boxingdouble d = (double)(int)o; // valid

 Note:

Boxing and unboxing degrade the performance. So, avoid using it. Use generics to avoid boxing and unboxing. For example, use List instead of ArrayList.