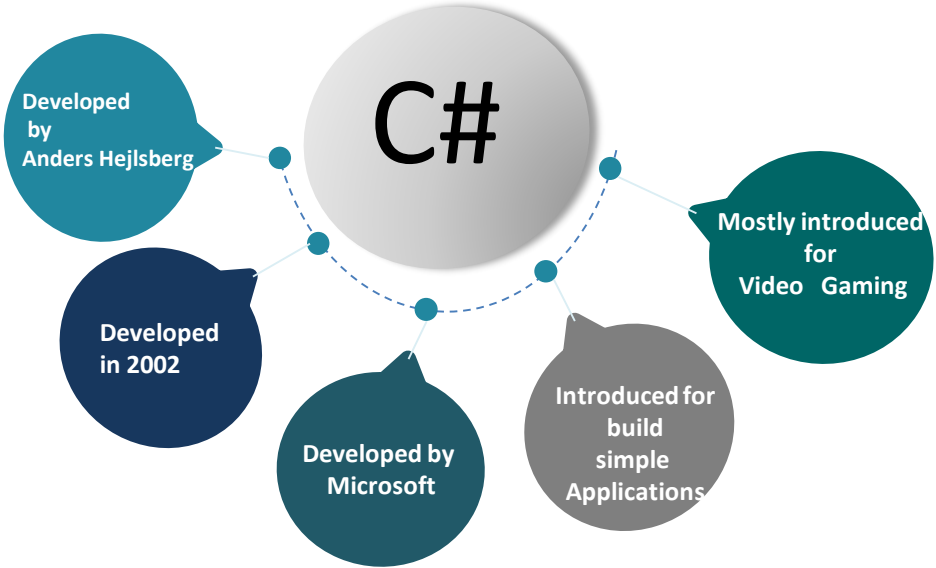
C++ Vs C#

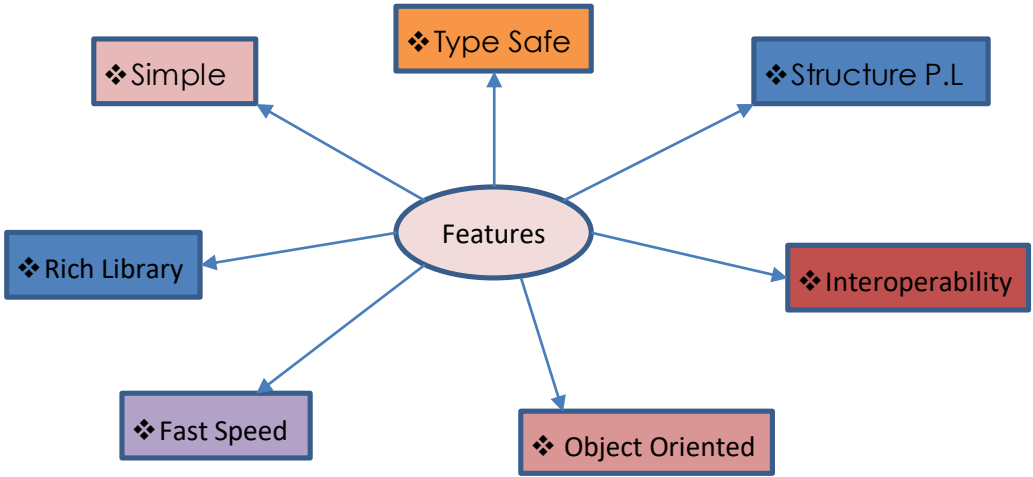
FortuneCloud

C++	C#
C++ compiles down to machine code.	C# 'compiles' down to CLR, which ASP.NET interprets (roughly) Edit - As pointed out in comments, it JIT compiles, not interprets.
In C++, you must handle memory manually.	Because C# runs in a virtual machine, memory management is handled automatically.
C++ support the multiple inheritance (of implementation).	C# does not (although it does have multiple inheritance of interface).
C++ includes a very powerful—and more complex and difficult to master—template meta language.	C# originally had nothing like it, with many people believing that the common type hierarchy obviated the need for such techniques.
In C++ you have to do your own memory management.	C# has a garbage collector.

History


C#
Developed by Anders Hejlsberg
Developed in 2002
Developed by Microsoft
Introduced for build simple Applications
Mostly introduced for Video Gaming

C# Features


❖Type Safe
❖Simple
❖Structure P.L
Features
❖Rich Library
❖Interoperability
❖Fast Speed
❖ Object Oriented

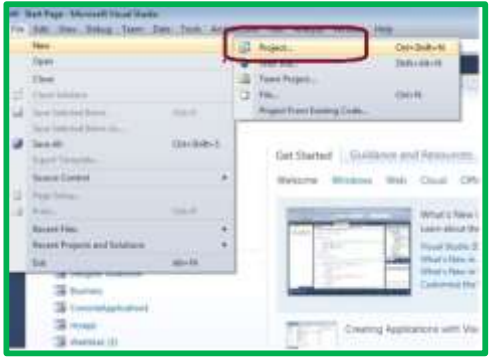
V S Installation

FortuneCloud

https://visualstudio.microsoft.com/thank-you-downloading-visual-studio/?sku=Community&rel=17

Create New Project

FortuneCloud
We Understand Technology



```
Using System;
```

Importing namespace section

```
Namespace namespacename
{
```

Namespace declaration section

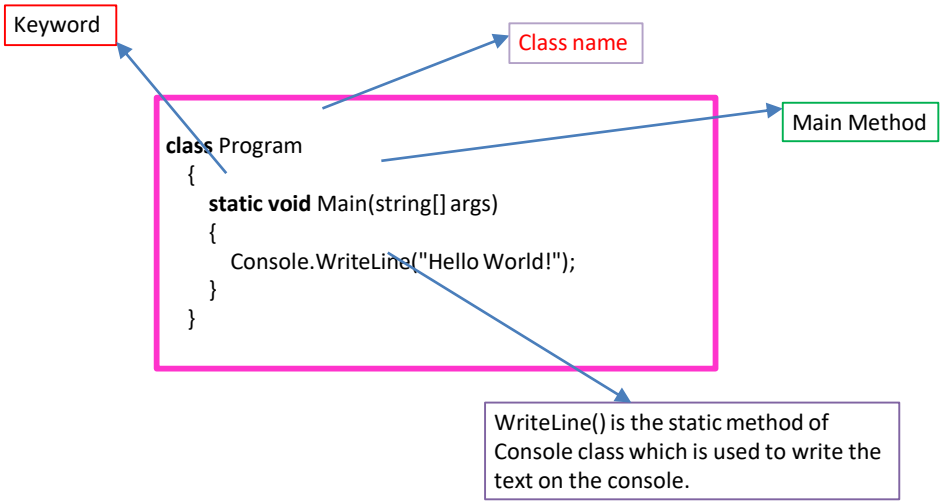
```
Class Class_name
{
```

Class declaration section

```
Static void Main(string args[])
{
    Class_name obj_name = new obj_name();
   Console.WriteLine(" Statements ");
}
}
}
```

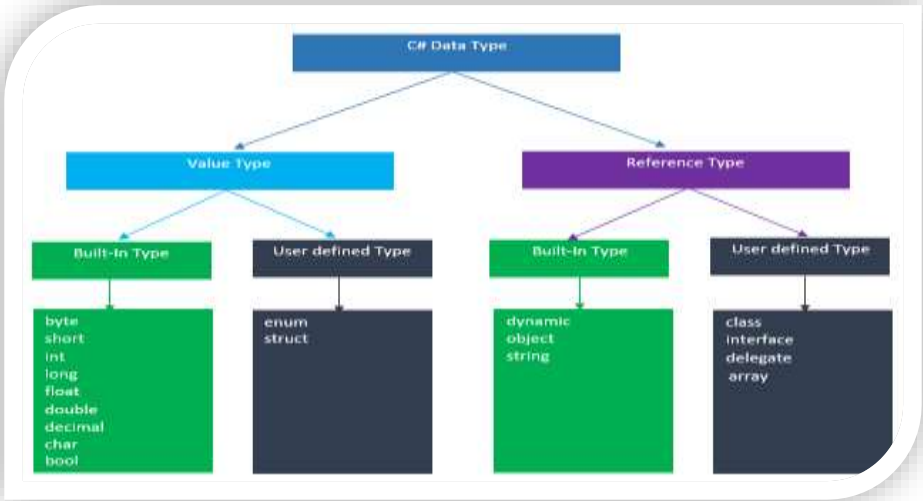
Main method section

First Project

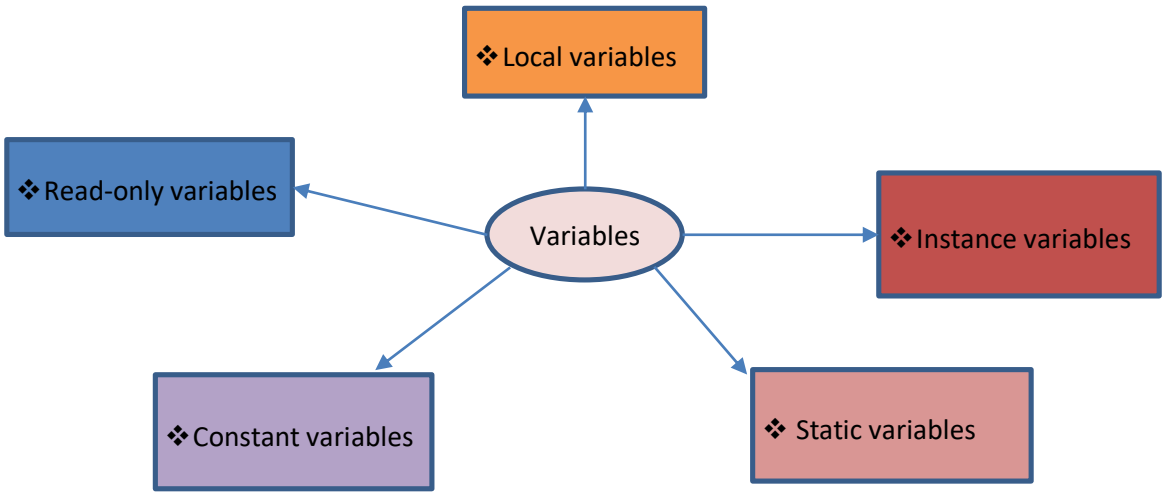


FortuneCloud
We Understand Technology

Data Type

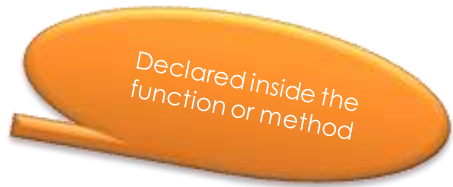


Variables in C#



Local Variable

```
class Student
{
   public void StudentAge()
   {

      // local variable age
      int age = 0;
      age = age + 10;
      Console.WriteLine("Student  age is : " + age);
   }
   public static void Main(String[] args)
   {
      Student obj = new Student();

      // calling the function
      obj.StudentAge();
   }
}
Output: Student age is: 10
```



FortuneCloud
We Understand Technology

Instance Variables

```
class Marks
{
   int engMarks;
   int mathsMarks;
   int phyMarks;

   public static void Main(String[] args)
   {
      Marks obj = new Marks();
      obj.engMarks = 90;
      obj.mathsMarks = 80;
      obj.phyMarks = 93;
      Console.WriteLine("English Marks:"+obj.engMarks);
      Console.WriteLine("Maths Marks:"+obj.mathsMarks);
     Console.WriteLine("Physics Marks:"+obj.phyMarks);
    }
}
```

- Declared outside the function or method
- Instance variables are created when class is created.

Static Variables

FortuneCloud
We Understand Technology

```
class Emp
{
   static double salary;
   static String name = "Aks";

   public static void Main(String[] args)
   {
      Emp.salary = 100000;

     Console.WriteLine(Emp.name + "'Average salary is:"+Emp.salary);
   }
}
```

- Declared outside the function or method using static keyword
- We can not access static variables using class object.

Constant Variables

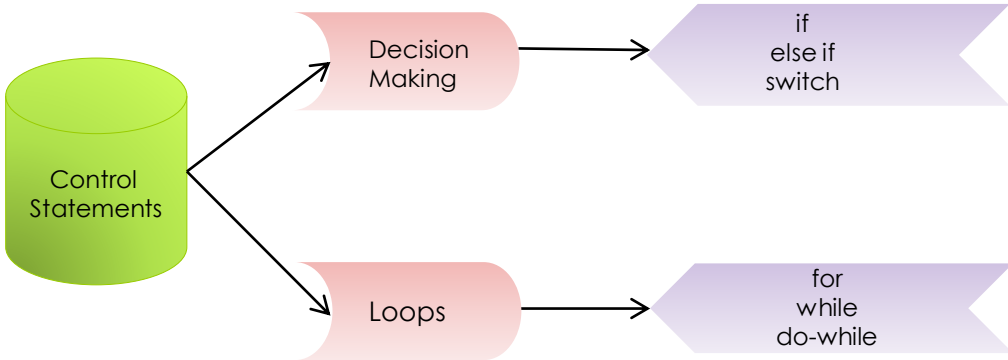
```
class Program {

    // must give value at the time of declaration
    const float max = 50;

    public static void Main()
    {
        Console.WriteLine("The value of max is = " + program.max);
    }
}
Output:
 The value of max is = 50
```

- Declared outside the function or method using Const keyword
- Constant variables are created at the time of declaration.

Read-only Variables

```
class Program
{
    readonly int k;

    public Program()
    {
        this.k = 90;
    }
    public static void Main()
{
   Program obj = new Program();

  Console.WriteLine("The value of k is = " + obj.k);
 }
}
```

- Declared outside as well as inside the function or method also using readonly keyword
- If we declare inside the function we have to use this keyword for reference of readonly variable.



if

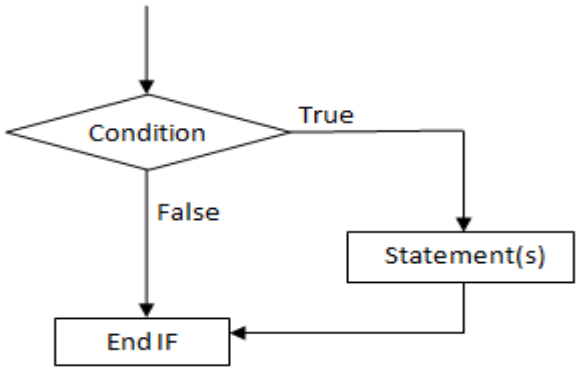


fig: Flowchart for if statement

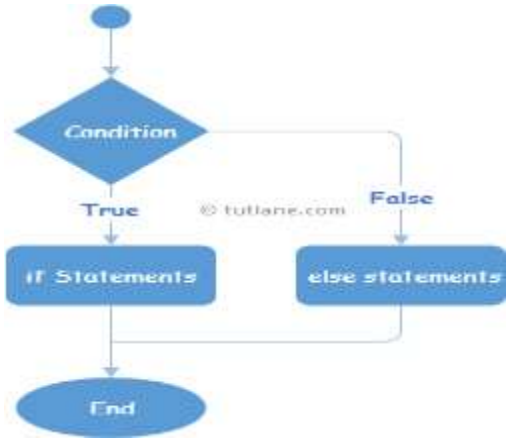
Syntax

```
if(condition)
{
    //code block
}
```

Example

```
class Program
 {
   static void Main(string[] args)
   {
      if(20>18)
      {
         Console.WriteLine("20 is greater than 18");
      }
    }
}
```

If-else


Condition
True
© tutlane.com
False
if Statements
else statements
End

Syntax

```
if(condition)
{
    //condition is true
}
Else
{
    //condition is false
}
```

Example:-

FortuneCloud
We Understand Technology

```
class Program
{
Public static void Main(string[] args)
{
        int num = 11;
        if (num % 2 == 0)
        {
           Console.WriteLine("It is even number");
        }
        else
        {
           Console.WriteLine("It is odd number");
        }
```

Switch

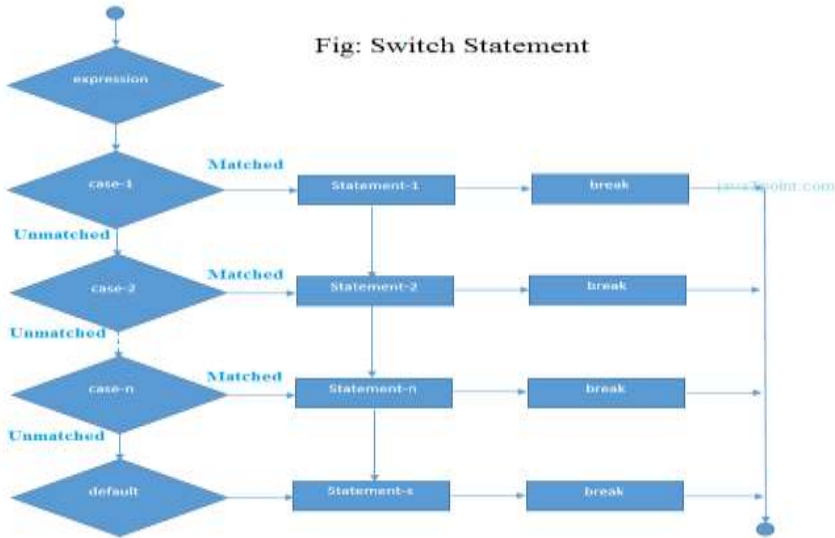


Fig: Switch Statement
expression
case-1
Matched
Statement-1
break
Unmatched
case-2
Matched
Statement-2
break
Unmatched
case-n
Matched
Statement-n
break
Unmatched
default
Statement-x
break

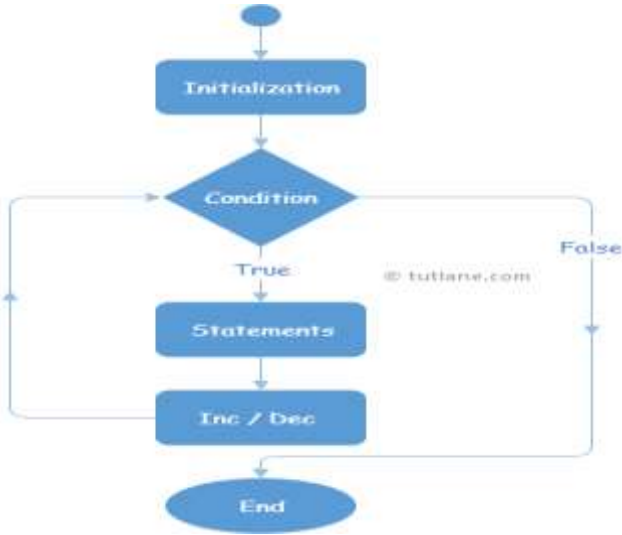
Syntax

```
switch(condition)
{
    case 1: //code block;
    break;
    case 2: //code block;
    break;
      .
      .
      .
     case n: //code block;
     break;
}
```

Example:-

```
    int day = 6;
switch (day)
 {
  case 1:
        Console.WriteLine("Monday");
   break;
   case 2:
       Console.WriteLine("Tuesday");
   break;
case 3:
      Console.WriteLine("Wednesday");
break;
case 4:
      Console.WriteLine("Thursday");
break;
case 5:
     Console.WriteLine("Friday");
break;
case 6:
      Console.WriteLine("Saturday");
break;
case 7:
       Console.WriteLine("Sunday");
 break;
}
```

For loop


Initialization
Condition
True
False
© tutlane.com
Statements
Inc / Dec
End

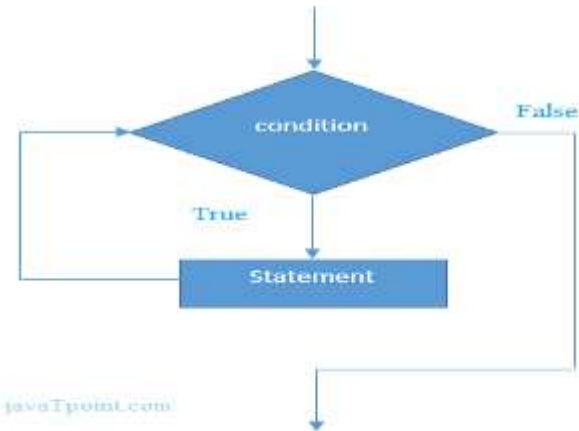
Syntax

```
for(loop variable; testing condition; Increment/Decrement)
{
    //condition is true
}
```

Example:-

```
class Program
 {
   static void Main(string[] args)
   {
     for (int i = 0; i < 5; i++)
     {
       Console.WriteLine(i);
     }
   }
}
```

While



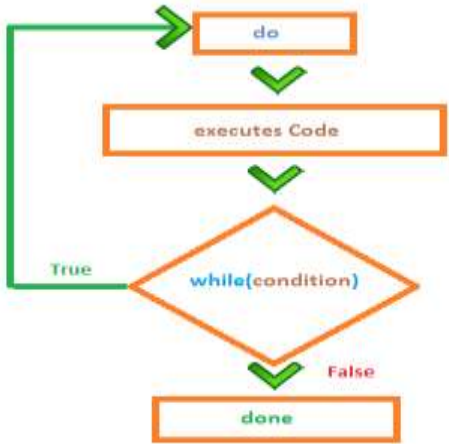
Syntax

```
while(condition)
{
    //loop block;
    // increment;
}
```

Example:-

```
class Program
 {
  static void Main(string[] args)
  {
   int i = 0;
   while (i < 5)
   {
    Console.WriteLine(i);
    i++;
   }
  }
 }
```

Do-while



Syntax

```
do
{
    //code block
}while(condition);
```

Example:-

FortuneCloud
We Understand Technology

```
class Program
 {
   static void Main(string[] args)
   {
     int i = 0;
     do
     {
       Console.WriteLine(i);
       i++;
     }while (i < 5);
   }
 }
```

Call By Value

FortuneCloud
We Understand Technology

```
class Program
  {
    public void Show(int val)
    {
      val *= val; // Manipulating value
      Console.WriteLine("Value inside the show function " + val);
      // No return statement
    }
    static void Main(string[] args)
    {
      int val = 50;
      Program p = new Program(); // Creating Object
      Console.WriteLine("Value before calling the function " + val);
      p.Show(val); // Calling Function by passing value
      Console.WriteLine("Value after calling the function " + val);


    }
  }
```

- Value type parameters are passed to copy of original value to the function rather than reference
- It does not modify the original value.

Call By Reference

FortuneCloud
We Understand Technology

```
class Program
   {
      public void Show(ref int val)
      {
          val *= val;

         Console.WriteLine("Value inside the show function "+val);
      }
       static void Main(string[] args)
      {
         int val = 50;

         Program p = new Program(); // Creating Object

         Console.WriteLine("Value before calling the function "+val);

         p.Show(ref val); // Calling Function by passing reference

          Console.WriteLine("Value after calling the function " + val);
      }
   }
}
```

- It provides ref keyword to pass arguments as reference-type
- It passes reference of arguments to the function.

Out parameters

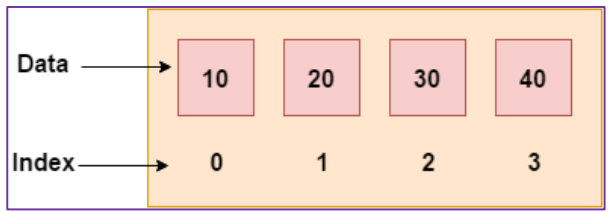
```
class Program
  {
     public void Show(out int val) // Out parameter
     {
        int square = 5;
        val = square;
        val *= val; // Manipulating value
     }

     static void Main(string[] args)
     {
        int val = 50;
        Program program = new Program(); // Creating Object
        Console.WriteLine("Value before passing out variable" + val);
        program.Show(out val); // Passing out argument
        Console.WriteLine("Value after recieving the out variable" + val)
;
     }
  }
}
```

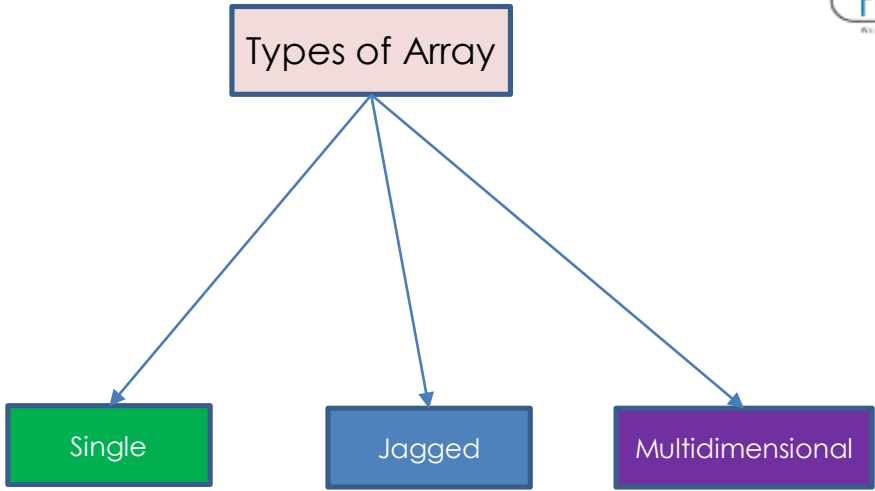
- It provides out keyword to pass arguments as out-type
- The main difference between ref and out is ref must initialize value before passing and out not needs to initialize value before passing.

Array:-

FortuneCloud
We Understand Technology

- An array is a group of like-typed variables that are referred to by a common name.
- Arrays are objects in C#, we can find their length using member length. This is different from C/C++ where we find length using sizeof operator.





Single Dimensional Array

FortuneCloud
We Understand Technology

There are 3 ways to initialize array at the time of declaration.

```
int[] arr = new int[5]; //creating array
```

```
int[] arr = new int[5]{ 10, 20, 30, 40, 50 };
```

```
int[] arr = new int[]{ 10, 20, 30, 40, 50 };
```

```
int[] arr = { 10, 20, 30, 40, 50 };
```

Example:-

FortuneCloud
We Understand Technology

```
using System;
public class ArrayExample
{
   public static void Main(string[] args)
   {
      int[] arr = { 10, 20, 30, 40, 50 };//Declaration
and Initialization of array

      //traversing array
      for (int i = 0; i < arr.Length; i++)
      {
         Console.WriteLine(arr[i]);
      }
   }
}
```

Multidimensional Arrays

FortuneCloud
We Understand Technology

- The multidimensional array is also known as rectangular arrays in C#.
- The data is stored in tabular form (row * column) which is also known as matrix.
- To create multidimensional array, we need to use comma inside the square brackets.

```
int[,] arr=new int[3,3];//declaration of 2D array
int[,,] arr=new int[3,3,3];//declaration of 3D array
```

```
int[,] arr = new int[3,3]= { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
```

```
int[,] arr = new int[,]{ { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
```

```
int[,] arr = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
```

Example

```
public static void Main(string[] args)
   {
      int[,] arr = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };//decl
and init

      //traversal
      for(int i=0;i<3;i++){
         for(int j=0;j<3;j++){
            Console.Write(arr[i,j]+" ");
         }
         Console.WriteLine();//new line at each row
      }
```

Jagged Arrays

FortuneCloud
We Understand Technology

- ✓ In C#, jagged array is also known as "array of arrays" because its elements are arrays.
- ✓ The element size of jagged array can be different.

Syntax:-

```
int[] [] arr = new int[2] [];
```

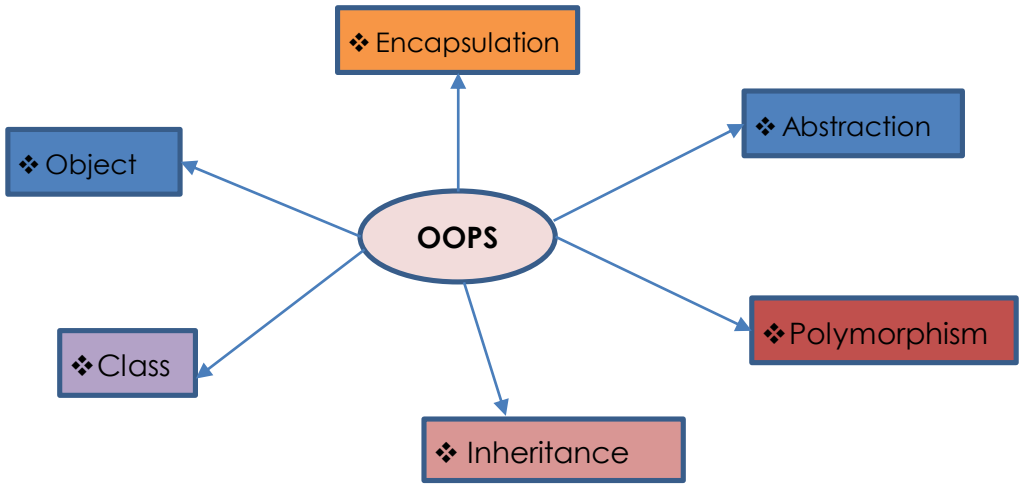
Size of an Array

dimensions of the array

Example

```
class Program
  {
     static void Main(string[] args)
     {
        // Declare the array of four elements:
        int[][] jaggedArray = new int[4][];
        // Initialize the elements:
        jaggedArray[0] = new int[2] { 7, 9 };
        jaggedArray[1] = new int[4] { 12, 42, 26, 38 };
        jaggedArray[2] = new int[6] { 3, 5, 7, 9, 11, 13 };
        jaggedArray[3] = new int[3] { 4, 6, 8 };
        // Display the array elements:
        for (int i = 0; i < jaggedArray.Length; i++)
        {
           System.Console.Write("Element({0}): ", i + 1);
           for (int j = 0; j < jaggedArray[i].Length; j++)
           {
              System.Console.Write(jaggedArray[i][j] + "\t");
           }
           System.Console.WriteLine();
        }
        Console.ReadLine();
     }
```

Object Oriented Programming



Class

FortuneCloud
We Understand Technology

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

```
class  class_name
{
  // data members of class
}
```

- Ex: class Car

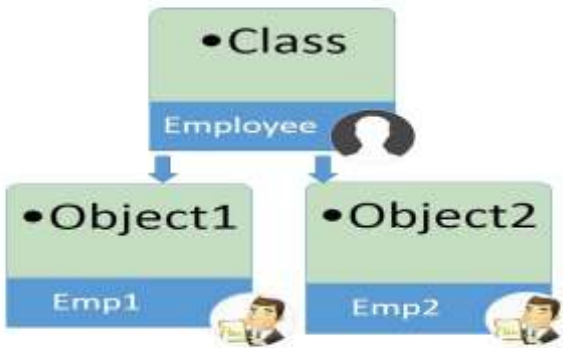
```
{
  string colour= "Red";
}
```

Object

- Any entity that has identity, state and behaviour.
- An object is an instance of a class.
- The object stores its identity and state in a variable / attribute and exposes its behaviour through method.
- Syntax:- class_name objec_tname = new class_name();
- Ex: class Car

```
{
  string colour= "Red";
  Public static void main(string args[])
  {
    Car mycar = new Car(mycar.colour);
  }
}
```

Real time Example of Class and Object



Abstraction

FortuneCloud
We Understand Technology

- Abstraction is "To represent the essential feature without representing the background details.“
- Abstraction can be achieved with either abstract classes or interfaces.

Example:-

```
abstract class Animal
 {
     public abstract void animalSound();
     public void sleep();
     {
       Console.WriteLine("Zzz");
     }
}
class Pig : Animal
{
     public override void animalSound()
     {
       Console.WriteLine("The pig says: wee wee");
      }
}
class Program
{
     static void Main(string[] args)
     {
       Pig myPig = new Pig();
       myPig.animalSound();
       myPig.sleep();
}
}
```

Encapsulation

FortuneCloud
We Understand Technology

- Encapsulation is the process of wrapping up the data members and member function into a single unit which can not be accessible by external entity.
- Encapsulation is like enclosing in a capsule. That is enclosing the related operations and data related to an object into that object.
- Example:

```
class Bag
{
  int book;
  int pen;
  Public void ReadBook();
}
```

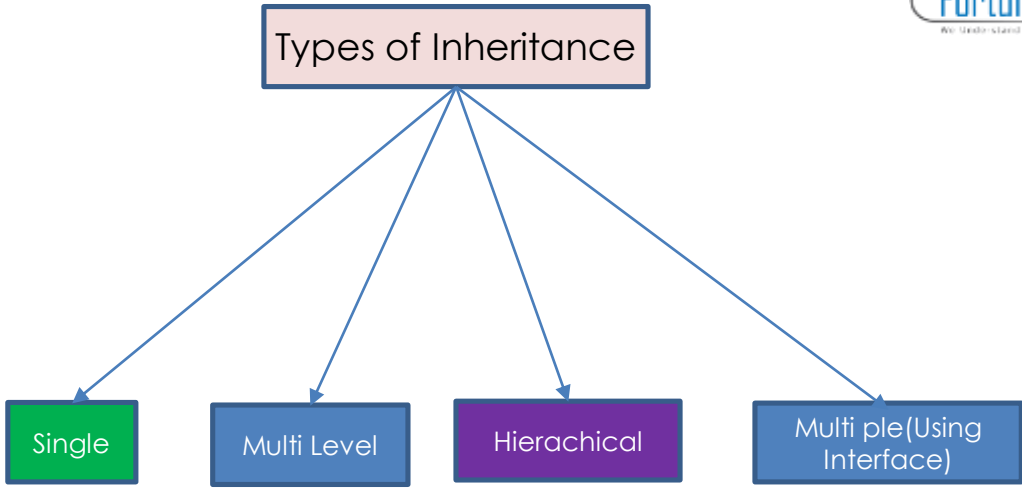
Inheritance

- When a class includes a property of another class it is known as inheritance.
- Inheritance is a process of object reusability.
- Example:

```
public class ParentClass
{
public ParentClass()
{
 Console.WriteLine("Parent  Constructor.");
}
public void print()
{
Console.WriteLine("I'm  a Parent Class.");
}
}
public class ChildClass: ParentClass
{
public ChildClass()
{
Console.WriteLine("Child  Constructor.");
}
public static void Main()
{
ChildClass child = new ChildClass();
child.print();
}
}
```

Example:-

```
public class ParentClass
{
      public ParentClass()
     {
        Console.WriteLine("Parent Constructor.");
     }
     public void print()
     {
        Console.WriteLine("I'm a Parent Class.");
     }
}
      public class ChildClass: ParentClass
     {
         public ChildClass()
        {
           Console.WriteLine("Child Constructor.");
         }
    public static void Main()
    {
        ChildClass child = new ChildClass();
        child.print();
     }
}
```



Single Inheritance

- It is the type of inheritance in which there is one base class and one derived class.
- Example:

```
Class Base
{
    public void accept()
    {
      Console.WriteLine("this is base class");
    }

}
Class derived
{
     public void display()
     {
        Console.WriteLine("this is derived class");
     }
   public static void Main()
   {
       Base obj = new Base();
       obj.accept();
       obj.display();
   }
}
```

Multilevel Inheritance

- When one class is derived from another derived class then this type of inheritance is called multilevel inheritance.
- Example

```
Class Base{
        public void accept()
        {
          Console.WriteLine("this is base class");
        }
}
Class Derived_A: Base{
                public void display()
                {
                  Console.WriteLine("this is Derived A class");
                }
}
Class Derived_B: Derived_A{
                        Public void displayAll()
                        {
                          Console.WriteLine("this is derived B calss");
}
    public static void Main(){
                        Derived_B obj = new Derived_B();
                         obj.accept();
                         obj.display();
                          obj.displayAll();
}
```

Hierarchical Inheritance

FortuneCloud
We Understand Technology

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

Example:-

```
Class Parent{
            public void AB()
           {
               Console.WriteLine("this is base class");
           }
}
Class Child_first: Base{
                        public void A()
                        {
                          Console.WriteLine("this is Child first");
                        }
}
Class Child_second{
                       public void B()
                       {
                          Console.WriteLine("this is child second");
                       }
                  public static void Main(string[] args)
                 {
                    Child_First obj = new Child_First()
                     obj.A();
                     obj.AB();
                    Child_second obj1 = new Child_second();
                    obj1.B();
                    obj1.AB();
                 }
}
```

Interface

- An interface looks like a class, but has no implementation. The only thing it contains are declarations of events, indexers, methods and/or properties. The reason interfaces only provide declarations is because they are inherited by structs and classes, that must provide an implementation for each interface member declared.

- To declare an interface, use *interface* keyword.

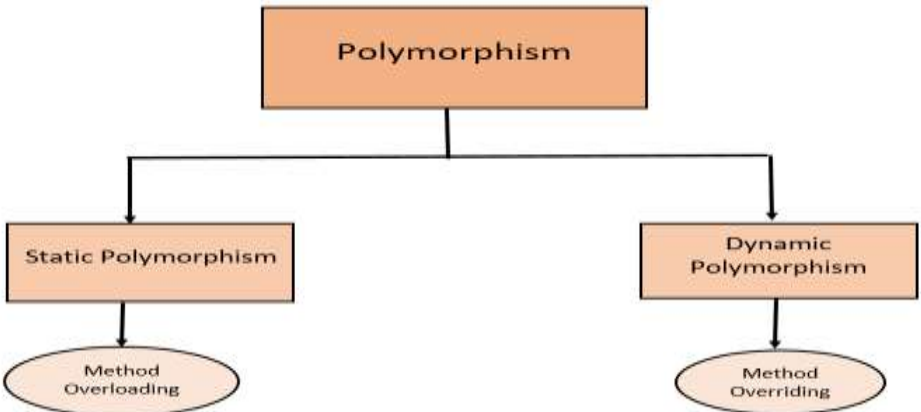
Example:-

```
interface inter1
  {
     // method having only declaration
     // not definition
     void display();
  }

  // A class that implements interface.
  class testClass : inter1
  {

     // providing the body part of function
     public void display()
     {
        Console.WriteLine("Fortune Cloud Technologies");
     }
     static void Main(string[] args)
     {
        testClass t = new testClass();

        // calling method
        t.display();
     }
  }
```

Polymorphism

- Polymorphism is a Greek word, meaning "one name many forms". In other words, one object has many forms or has one name with multiple functionalities. "Poly" means many and "morph" means forms. Polymorphism provides the ability to a class to have multiple implementations with the same name.

Types of Polymorphism



Method Overloading

FortuneCloud
We Understand Technology

- Method Overloading is a type of polymorphism. It has several names like "Compile Time Polymorphism" or "Static Polymorphism" and sometimes it is called "Early Binding".
- Method Overloading means creating multiple methods in a class with same names but different signatures (Parameters). It permits a class, struct, or interface to declare multiple methods with the same name with unique signatures.

Example:-

```
class Program
  {
     public int Add(int a, int b)
     {
        int sum = a + b;
        return sum;
     }

     // adding three integer values.
     public int Add(int a, int b, int c)
     {
        int sum = a + b + c;
        return sum;
     }
     static void Main(string[] args)
     {
        Program ob = new Program();
        int sum1 = ob.Add(1, 2);
        Console.WriteLine("sum of the two "
                          + "integer value : " + sum1);

        int sum2 = ob.Add(1, 2, 3);
        Console.WriteLine("sum of the three "
                          + "integer value : " + sum2);
     }
  }
```

Method Overriding

- Method Overriding is a type of polymorphism. It has several names like "Run Time Polymorphism" or "Dynamic Polymorphism" and sometime it is called "Late Binding".
- Method Overriding means having two methods with same name and same signatures [parameters], one should be in the base class and other method should be in a derived class [child class]. You can override the functionality of a base class method to create a same name method with same signature in a derived class. You can achieve method overriding using inheritance. Virtual and Override keywords are used to achieve method overriding.

Example:-


```
class baseClass
   {
      public virtual void Greetings()
      {
         Console.WriteLine("baseClass Saying Hello!");
      }
   }
   class subClass : baseClass
   {
      public override void Greetings()
      {
         base.Greetings();
         Console.WriteLine("subClass Saying Hello!");
      }
   }
   class Program
   {
      static void Main(string[] args)
      {
         subClass obj1 = new subClass();
         obj1.Greetings();
         Console.ReadLine();
      }
```

Constructor

FortuneCloud

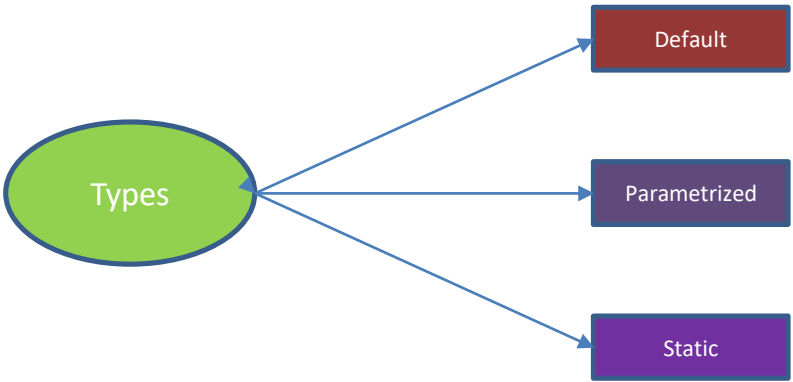
- Constructors are a particular type of method associated with a class and gets automatically invoked when the classes instance (i.e., objects) are created.
- The name of the constructors must have to be the same as that of the class's name in which will resides.
- The main use of constructors is to initialize the private fields of the class while creating an instance for the class.

Rules for Constructor

- A constructor doesn't have any return type, not even void.
- A static constructor can not be a parameterized constructor.
- Within a class you can create only one static constructor.
- The number of constructors can be any within a class.
- Constructors can contain access modifiers along with it.
- A class can have any number of constructors.

Example:-


```
class Program
{
 ..... // Constructor
public Program()
{
}
.....
 // an object is created of Geek class,
// So above constructor is called
Program obj = new Program();
 obj.Program
}
}
```



Default Constructor Example

FortuneCloud
We Understand Technology

- A constructor without any parameters is called a default constructor; in other words, this type of constructor does not take parameters.
- Example

```
class addition
{
   int a, b;
   public addition() //default contructor
   {
      a = 100;
      b = 175;
   }
      public static void Main()
      {
         addition obj = new addition(); //an object is created , constructor is called
         Console.WriteLine(obj.a);
         Console.WriteLine(obj.b);
         Console.Read();
       }
```

Parameterize Constructor Example

- A constructor with at least one parameter is called a parameterized constructor. The advantage of a parameterized constructor is that you can initialize each instance of the class with a different value.
- Example

```
class paraconstrctor
{
    public int a, b;
    public paraconstrctor(int x, int y) // decalaring Paremetrized Constructor with ing x,y parameter
    {
      a = x;
      b = y;
    }
}
    class MainClass { static void Main()
    {
  paraconstrctor v = new paraconstrctor(100, 175); // Creating object of Parameterized  Constructor and ing
values
 Console.WriteLine("\t");
Console.WriteLine("value of a=" + v.a );
Console.WriteLine("value of b=" + v.b);
Console.Read();
}
```

Copy Constructor Example

FortuneCloud
We Understand Technology

- The constructor which creates an object by copying variables from another object is called a copy constructor. The purpose of a copy constructor is to initialize a new instance to the values of an existing instance.

Example:-

```
class employee
{
private string name;
private int age;
public employee(employee emp) // declaring Copy constructor.
{
name = emp.name;
age = emp.age;
}
public employee(string name, int age)
{
this.name = name;
this.age = age;
}
public string Details()
{
return " The age of " + name +" is "+ age.ToString();
 }
 }
 }
class empdetail
{
static void Main()
{
employee emp1 = new employee("Vithal", 23);
employee emp2 = new employee(emp1);
Console.WriteLine(emp2.Details);
Console.ReadLine();
}
```

Static Constructor Example

FortuneCloud
We Understand Technology

A static constructor is used to initialize static variables of the class and to perform a particular action only once.
It is invoked before any static member of the class is accessed.
A static constructor does not take any parameters and does not use any access modifiers because it is invoked directly by the CLR instead of the object.

Example:-

```
public class employee
{
static employee()
{
Console.WriteLine("The static constructor ");
}
public static void Salary()
{
Console.WriteLine();
Console.WriteLine("The Salary method");
}
}
class details
{
static void Main()
{
 Console.WriteLine();
employee.Salary();
Console.ReadLine();
}
}
```

Private Constructor

FortuneCloud
We Understand Technology

- When a constructor is created with a private specifies,it is not possible for other classes to derive from this class, neither is it possible to create an instance of this class. They are usually used in classes that contain static members only.
- One use of a private constructor is when we have only static members.
- It provides an implementation of a singleton class pattern
- Once we provide a constructor that is either private or public or any, the compiler will not add the parameter-less public constructor to the class.

Example:-

```
public class Counter
  {
    private Counter()   //private constrctor declaration
    {
    }
    public static int currentview;
    public static int visitedCount()
    {
      return ++currentview;
    }
  }
  class viewCountedetails
  {
    static void Main(string[] args)
    {
      // Counter aCounter = new Counter();
      Console.WriteLine();
      Counter.currentview = 500;
      Counter.visitedCount();
      Console.WriteLine("Now the view count is: {0}", Counter.currentview);
      Console.ReadLine();
   }
```

Destructor

- A destructor works opposite to constructor, It destructs the objects of classes. It can be defined only once in a class. Like constructors, it is invoked automatically.

Example:-

```
public class Employee
  {
     public Employee()
     {
        Console.WriteLine("Constructor Invoked");
     }
     ~Employee()
     {
        Console.WriteLine("Destructor Invoked");
     }
  }
  class TestEmployee
  {
     static void Main(string[] args)
     {
        Employee e1 = new Employee();
        Employee e2 = new Employee();

     }
  }
```

This Keyword

FortuneCloud
We Understand Technology

- The C# “this” keyword represents the “this” pointer of a class or a stuct. The this pointer represents the current instance of a class or stuct.
- The this pointer is a pointer accessible only within the nonstatic methods of a class or struct. It points to the object for which the method is called. Static members of a class or struct do not have a this pointer. The this pointer is also used on hidden fields to separate fields with the method parameters with the same names.
- In our code, there are times when we need to pass an instance of the current class or struct to the outside classes and their methods. This is where the "this" pointer of a class or struct is used.

Example:-

```
class Student
  {
     public int id, age;
     public String name, subject;

     public Student(int id, String name, int age, String subject)
     {
        this.id = id;
        this.name = name;
        this.subject = subject;
        this.age = age;
     }

     public void showInfo()
     {
        Console.WriteLine(id + " " + name + " " + age + " " + subject);
     }
  }

  class StudentDetails
  {
     static void Main(string[] args)
     {
        Student std1 = new Student(001, "Jack", 23, "Maths");
        Student std2 = new Student(002, "Harry", 27, "Science");
        Student std3 = new Student(003, "Steve", 23, "Programming");
        Student std4 = new Student(004, "David", 27, "English");
        std1.showInfo();
        std2.showInfo();
        std3.showInfo();
        std4.showInfo();
     }
```

Static Class

FortuneCloud
We Understand Technology

- A static class is a class that cannot be instantiated. The main purpose of using static classes in C# is to provide blueprints of its inherited classes. Static classes are created using the static keyword in C# and .NET. A static class can contain static members only. You can't create an object for the static class.

Example:-


```
static class Author
 {
    public static string A_name = "Ankita";
    public static string L_name = "CSharp";
    public static int T_no = 84;
       public static void details()
       {
        Console.WriteLine("The details of Author is:");
        }
}
 public class GFG
{
    static public void Main()
    {
        Author.details();
        // Accessing the static data members of Author
        Console.WriteLine("Author name : {0} "+Author.A_name);
        Console.WriteLine("Language : {0} "+Author.L_name);
   Console.WriteLine("Total number of articles : {0} "+Author.T_no);
    }
}
```

Sealed Class

- Sealed classes are used to restrict the inheritance feature of object oriented programming. Once a class is defined as a **sealed class**, this class cannot be inherited.
- If you create a sealed method, it cannot be overridden.
- Syntax:

```
sealed class class_name
{
// data members
// methods
. . .
}
```

Example:-

```
sealed class SealedClass
{
    // Calling Function
    public int Add(int a, int b)
    {
        return a + b;
    }
}
 class Program
{
    // Main Method
    static void Main(string[] args)
    {
        // Creating an object of Sealed Class
        SealedClass slc = new SealedClass();

        // Performing Addition operation
        int total = slc.Add(6, 4);
        Console.WriteLine("Total = " + total.ToString());
    }
}
```

String

- Strings are used for storing text.
- A string variable contains a collection of characters surrounded by double quotes.
- Syntax
  String Vriable_name = “string_name”;
- Example
  String Greeting=“Hello”;

Operations on String

❖ String Length

```
class Program
 {
static void Main(string[] args)
{
string txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
Console.WriteLine("The length of the txt string is: " + txt.Length);
}
```

Operations on String

❖ String Concatenation

```
class Program
{
static void Main(string[] args)
{
string firstName = "John ";
string lastName = "Doe";
string name = firstName + lastName;
Console.WriteLine("Concatenated String ="+name);
}
```

❖ You can also use the string.Concat() method to concatenate two strings:

```
class Program
{
static void Main(string[] args)
{
string firstName = "John ";
string lastName = "Doe";
string name = string.Concat(firstName, lastName);
Console.WriteLine("Concatenated String ="+name);
}
```

Operations on String

- String methods ToUpper() and ToLower(), which returns a copy of the string converted to uppercase or lowercase:

```
class Program
{
static void Main(string[] args)
{
string txt = "Hello World";
Console.WriteLine(txt.ToUpper()); // Outputs "HELLO WORLD"
Console.WriteLine(txt.ToLower()); // Outputs "hello world"
}
```

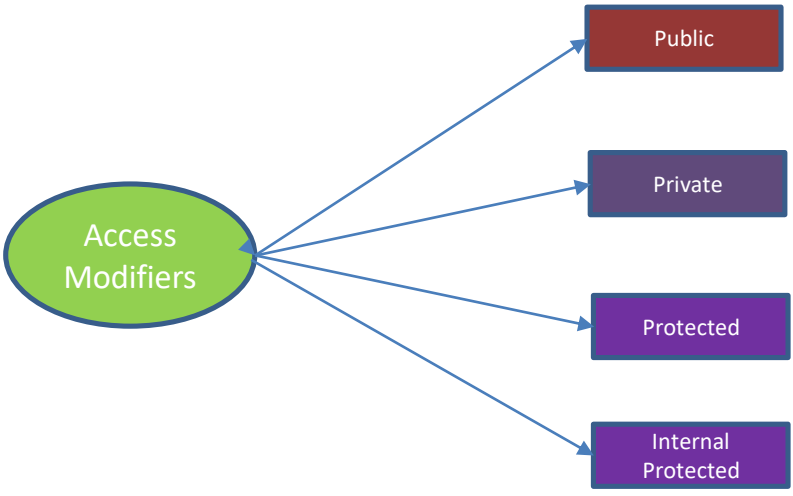
Operations on String

- Access String: You can access the characters in a string by referring to its index number inside square brackets [].

```
class Program
{
static void Main(string[] args)
{
string myString = "Hello";
Console.WriteLine(myString[0]);
}
```

- You can also find the index position of a specific character in a string, by using the IndexOf() method:

```
class Program
{
static void Main(string[] args)
{
string myString = "Hello";
Console.WriteLine(myString.IndexOf("e"));
}
```



Public

- Access is granted to the entire program. This means that another method or another assembly which contains the class reference can access these members or types. This access modifier has the most permissive access level in comparison to all other access modifiers.
- Example:

```
class Car
{
public string model = "Mustang";
}
class Program
{
static void Main(string[] args)
{
Car myObj = new Car();
Console.WriteLine(myObj.model);
}
}
```

Private

- Access is only granted to the containing class. Any other class inside the current or another assembly is not granted access to these members.
- Example:

```
class Car
{
private string model = "Mustang";
static void Main(string[] args)
{
Car myObj = new Car();
Console.WriteLine(myObj.model);
}
}
```

Protected

- Access is limited to the class that contains the member and derived types of this class. The code is accessible within the same class, or in a class that is inherited from that class.
- Example:

```
class X {
protected int x;
public X()
{
x = 10;
}
}
class Y : X {
public int getX()
{
return x;
 }
}
class Program {

   static void Main(string[] args)
   {
      X obj1 = new X();
      Y obj2 = new Y();

      // Displaying the value of x
      Console.WriteLine("Value of x is : {0}"+ obj2.getX());
   }
}
```

Internal

- Access is limited to only the current Assembly, that is any class or type declared as internal is accessible anywhere inside the same namespace. The code is only accessible within its own assembly, but not from another assembly.
- Example:

```
internal class Complex {
   int real;
    int img;
    public void setData(int r, int i)
    {
        real = r;
        img = i;
    }
    public void displayData()
    {
        Console.WriteLine("Real = {0}"+ real);
        Console.WriteLine("Imaginary = {0}"+ img);
    }
}
 class Program {
    static void Main(string[] args)
    {
        Complex c = new Complex();
        c.setData(2, 1);
        c.displayData();
    }
}
```

Exception Handling

- An exception is a problem that arises during the execution of a program. Exceptions provide a way to transfer control from one part of a program to another.
- **try** − A try block identifies a block of code for which particular exceptions is activated. It is followed by one or more catch blocks.
- **catch** − A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.
- **finally** − The finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised or not.
- **throw** − A program throws an exception when a problem shows up. This is done using a throw keyword.

Exception Handling

❖ Syntax:

```
try
{
// statements causing exception
}
catch( ExceptionName e1 )
{
// error handling code
}
catch( ExceptionName e2 )
{
 // error handling code
}
catch( ExceptionName eN )
{
// error handling code
}
finally
{
 // statements to be executed
}
```

Example 1:-

```
class Program
 {
  static void Main(string[] args)
  {
   try
   {
    int[] myNumbers = {1, 2, 3};
    Console.WriteLine(myNumbers[10]);
   }
   catch (Exception e)
   {
    Console.WriteLine("Something went wrong.");
   }
   finally
   {
    Console.WriteLine("The 'try catch' is finished.");
   }
  }
 }
```

Example 2:-


```
class DivNumbers
{
    int result;
    DivNumbers()
    {
      result = 0;
    }
      public void division(int num1, int num2)
      {
         try
         {
           result = num1 / num2;
         }
        catch (DivideByZeroException e)
        {
          Console.WriteLine("Exception caught: {0}", e);
        }
        Finally
        {
           Console.WriteLine("Result: {0}", result);
        }
}
static void Main(string[] args)
{
DivNumbers d = new DivNumbers();
d.division(25, 0);
Console.ReadKey();
}
```

Checked Exception

- The checked keyword is used to explicitly check overflow and conversion of integral type values at compile time.
- Example:

```
class Program
  {
    static void Main(string[] args)
    {
      checked
      {
        int val = int.MaxValue;
        Console.WriteLine(val + 2);
      }
    }
```

Unchecked Exception

- The Unchecked keyword ignores the integral type arithmetic exceptions. It does not check explicitly and produce result that may be truncated or wrong.
- Example:

```
class Program
  {
    static void Main(string[] args)
    {
      unchecked
      {
        int val = int.MaxValue;
        Console.WriteLine(val + 2);
      }
    }
```

File IO System

- 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**.
- **Objectives**

1. Using the File class for reading and writing data.
2. Using the File and FileInfo class to manipulate files.
3. Using the DirectoryInfo and Directory classes to manipulate directories.

Write To a File and Read It

FortuneCloud

- we use the WriteAllText() method to create a file named "filename.txt" and write some content to it. Then we use the ReadAllText() method to read the contents of the file:
- Example:

```
class Program
{
static void Main(string[] args)
{
string writeText = "Hello World!";
File.WriteAllText("filename.txt", writeText);
readText = File.ReadAllText("filename.txt");
Console.WriteLine(readText);
}
}
```

Stream Reader

FortuneCloud
We Understand Technology

- StreamReader is used to read characters to a stream in a specified encoding. StreamReader.Read method reads the next character or next set of characters from the input stream. StreamReader is inherited from TextReader that provides methods to read a character, block, line, or all content.

Example:-

```
using System;
using System.IO;
using System.Text;
class Program
{
static void Main(string[] args)
{
string fileName = @"C:\ Temp\CSharpAuthors.txt";
try
{
using (StreamReader reader = new StreamReader(fileName))
{
string line;
while ((line = reader.ReadLine()) != null)
{
Console.WriteLine(line);
}
}
}
catch (Exception exp)
{
Console.WriteLine(exp.Message);
}
Console.ReadKey();
}
}
```

Stream Writer

FortuneCloud
We Understand Technology

- StreamWriter class in C# writes characters to a stream in a specified encoding. StreamWriter.Write() method is responsible for writing text to a stream. StreamWriter class is inherited from TextWriter class that provides methods to write an object to a string, write strings to a file, or to serialize XML.

Example:-

```
using System;
using System.IO;
using System.Text;
class Program
{
  static void Main(string[] args)
  {
    string fileName = @"C:\Temp\CSharpAuthors.txt";
    FileStream stream = null;
    try
    {
      stream = new FileStream(fileName, FileMode.OpenOrCreate);
      using (StreamWriter writer = new StreamWriter(stream, Encoding.UTF8 ))
      {
        writer.WriteLine("C# Corner Authors");
        writer.WriteLine("==================");
        writer.WriteLine("Monica Rathbun");
        writer.WriteLine("Vidya Agarwal");
        writer.WriteLine("Mahesh Chand");
        writer.WriteLine("Vijay Anand");
        writer.WriteLine("Jignesh Trivedi");
      }
    }
finally
  {
    if (stream != null)
    stream.Dispose();
  }
  string readText = File.ReadAllText(fileName);
  Console.WriteLine(readText);
  Console.ReadKey();
  }
```

Text Reader

```
class Program
    {
static void Main(string[] args)
        {
            using (TextReader txtR = File.OpenText("d:\\textFile.txt")) {

             String data = txtR.ReadToEnd();
             Console.WriteLine(data);
             }
             Console.ReadLine();
           }
```

Text Writer

FortuneCloud
We Understand Technology

```
class Program
    {
static void Main(string[] args)
          {
using (TextWriter writer = File.CreateText("d:\\textFile.txt")) {

writer.WriteLine("The first line with text writer");
             }
Console.ReadLine();
          }
    }
```

Binary Writer

FortuneCloud
We Understand Technology

C# BinaryWriter class is used to write binary information into stream. It is found in System.IO namespace. It also supports writing string in specific encoding.

Example:-

```
class Program
   {
      static void Main(string[] args)
      {
         string fileName = "e:\\binaryfile.dat";
         using (BinaryWriter writer = new BinaryWriter(File.Open(fileName, FileMode.Create)))

         {
            writer.Write(2.5);
            writer.Write("this is string data");
            writer.Write(true);
         }
         Console.WriteLine("Data written successfully...");
      }
```

Binary Reader

FortuneCloud
We Understand Technology

C# BinaryReader class is used to read binary information from stream. It is found in System.IO namespace. It also supports reading string in specific encoding.

Example:-

```
class Program
  {
    static void Main(string[] args)
    {
      WriteBinaryFile();
      ReadBinaryFile();
      Console.ReadKey();
    }
    static void WriteBinaryFile()
    {
      using (BinaryWriter writer = new BinaryWriter(File.Open("e:\\binaryfile.dat", FileMode.Create)))
      {

        writer.Write(12.5);
        writer.Write("this is string data");
        writer.Write(true);
      }
    }
    static void ReadBinaryFile()
    {
      using (BinaryReader reader = new BinaryReader(File.Open("e:\\binaryfile.dat", FileMode.Open)))
      {
        Console.WriteLine("Double Value : " + reader.ReadDouble());
        Console.WriteLine("String Value : " + reader.ReadString());
        Console.WriteLine("Boolean Value : " + reader.ReadBoolean());
      }
    }
```

String Reader and Writer

FortuneCloud
We Understand Technology

- StringReader class is used to read data written by the StringWriter class. It is subclass of TextReader class. It enables us to read a string synchronously or asynchronously. It provides constructors and methods to perform read operations.

Example:-

```
class Program
   {
      static void Main(string[] args)
      {
         StringWriter str = new StringWriter();
         str.WriteLine("Hello, this message is read by StringReader class");
         str.Close();
         // Creating StringReader instance and passing StringWriter
         StringReader reader = new StringReader(str.ToString());
         // Reading data
         while (reader.Peek() > -1)
         {
            Console.WriteLine(reader.ReadLine());
         }
      }
```

FileInfo Class

FortuneCloud
We Understand Technology

- The FileInfo class is used to deal with file and its operations in C#. It provides properties and methods that are used to create, delete and read file. It uses StreamWriter class to write data to the file. It is a part of System.IO namespace.
- FileInfo Example: Creating a File

```
class Program
  {
    static void Main(string[] args)
    {
      try
      {
        // Specifying file location
        string loc = "F:\\abc.txt";
        // Creating FileInfo instance
        FileInfo file = new FileInfo(loc);
        // Creating an empty file
        file.Create();
        Console.WriteLine("File is created Successfuly");
      }catch(IOException e)
      {
        Console.WriteLine("Something went wrong: "+e);
      }
```

❖ FileInfo Example: writing to the file

```
class Program
  {
    static void Main(string[] args)
    {
      try
      {
        // Specifying file location
        string loc = "F:\Cravita\abc.txt";
        // Creating FileInfo instance
        FileInfo file = new FileInfo(loc);
        // Creating an file instance to write
        StreamWriter sw = file.CreateText();
        // Writing to the file
        sw.WriteLine("This text is written to the file by using StreamWriter class.");
        sw.Close();
      }catch(IOException e)
      {
        Console.WriteLine("Something went wrong: "+e);
      }
```

❖ FileInfo Example: Reading text from the file

```
class Program
  {
     static void Main(string[] args)
     {
        try
        {
           // Specifying file to read
           string loc = "F:\\abc.txt";
           // Creating FileInfo instance
           FileInfo file = new FileInfo(loc);
           // Opening file to read
           StreamReader sr = file.OpenText();
           string data = "";
           while ((data = sr.ReadLine()) != null)
           {
              Console.WriteLine(data);
           }
        }
        catch (IOException e)
        {
           Console.WriteLine("Something went wrong: " + e);
        }
```

DirectoryInfo Class

FortuneCloud
We Understand Technology

- DirectoryInfo class is a part of System.IO namespace. It is used to create, delete and move directory. It provides methods to perform operations related to directory and subdirectory. It is a sealed class so, we cannot inherit it.

DirectoryInfo Example: Creating Directory


```
class Program
  {
    static void Main(string[] args)
    {
      // Provide directory name with complete location.
      DirectoryInfo directory = new DirectoryInfo(@"F:\javatpoint");
      try
      {
        // Check, directory exist or not.
        if (directory.Exists)
        {
          Console.WriteLine("Directory already exist.");
          return;
        }
        // Creating a new directory.
        directory.Create();
        Console.WriteLine("The directory is created successfully.");
      }
      catch (Exception e)
      {
        Console.WriteLine("Directory not created: {0}", e.ToString());
      }
```

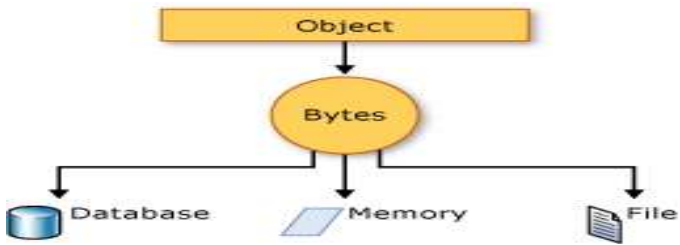
DirectoryInfo Example: Deleting Directory


```
class Program
  {
     static void Main(string[] args)
     {
        // Providing directory name with complete location.
        DirectoryInfo directory = new DirectoryInfo(@"F:\javatpoint");
        try
        {
           // Deleting directory
           directory.Delete();
           Console.WriteLine("The directory is deleted successfully.");
        }
        catch (Exception e)
        {
           Console.WriteLine("Something went wrong: {0}", e.ToString());
        }
```

Serialization

FortuneCloud

- Serialization is the process of converting object into byte stream so that it can be saved to memory, file or database. The reverse process of serialization is called deserialization.
- Serialization is internally used in remote applications.



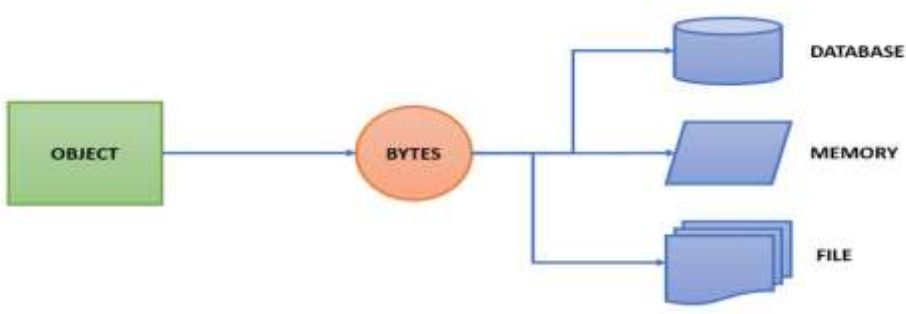
Example


```
using System;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
[Serializable]
class Student
{
   int rollno;
   string name;
   public Student(int rollno, string name)
   {
      this.rollno = rollno;
      this.name = name;
   }
}
public class SerializeExample
{
   public static void Main(string[] args)
   {
      FileStream stream = new FileStream("e:\\sss.txt", FileMode.OpenOrCreate);
      BinaryFormatter formatter=new BinaryFormatter();

      Student s = new Student(101, "sonoo");
      formatter.Serialize(stream, s);

      stream.Close();
   }
```

Deserialization

FortuneCloud

- Deserialization is the reverse process of serialization. It means you can read the object from byte stream. Here, we are going to use **BinaryFormatter.Deserialize(stream)** method to deserialize the stream.



Example

FortuneCloud

```
class Student
{
   public int rollno;
   public string name;
   public Student(int rollno, string name)
   {
      this.rollno = rollno;
      this.name = name;
   }
}
public class DeserializeExample
{
   public static void Main(string[] args)
   {
      FileStream stream = new FileStream("e:\\sss.txt", FileMode.OpenOrCreate);
      BinaryFormatter formatter=new BinaryFormatter();

      Student s=(Student)formatter.Deserialize(stream);
      Console.WriteLine("Rollno: " + s.rollno);
      Console.WriteLine("Name: " + s.name);

      stream.Close();
   }
```

Collections

FortuneCloud
We Understand Technology

- Collection types are designed to store, manage and manipulate similar data more efficiently. Data manipulation includes adding, removing, finding, and inserting data in the collection.

1. Adding and inserting items to a collection
2. Removing items from a collection
3. Finding, sorting, searching items
4. Replacing items
5. Copy and clone collections and items
6. Capacity and Count properties to find the capacity of the collection and number of items in the collection

List<T>

FortuneCloud

- List<T> class is used to store and fetch elements. It can have duplicate elements. It is found in System.Collections.Generic namespace.
- Example:

```
public class ListExample
{
   public static void Main(string[] args)
   {
      // Create a list of strings
      var names = new List<string>();
      names.Add("Sonoo Jaiswal");
      names.Add("Ankit");
      names.Add("Peter");
      names.Add("Irfan");

      // Iterate list element using foreach loop
      foreach (var name in names)
      {
         Console.WriteLine(name);
      }
```

HashSet<T>

FortuneCloud
We Understand Technology

- HashSet class can be used to store, remove or view elements. It does not store duplicate elements. It is suggested to use HashSet class if you have to store only unique elements. It is found in System.Collections.Generic namespace.
- Example:

```
public class HashSetExample
{
   public static void Main(string[] args)
   {
      // Create a set of strings
      var names = new HashSet<string>();
      names.Add("Sonoo");
      names.Add("Ankit");
      names.Add("Peter");
      names.Add("Irfan");
      names.Add("Ankit");//will not be added

      // Iterate HashSet elements using foreach loop
      foreach (var name in names)
      {
         Console.WriteLine(name);
      }
```

SortedSet<T>

- SortedSet class can be used to store, remove or view elements. It maintains ascending order and does not store duplicate elements. It is suggested to use SortedSet class if you have to store unique elements and maintain ascending order
- Example:

```
public class SortedSetExample
{
   public static void Main(string[] args)
   {
      // Create a set of strings
      var names = new SortedSet<string>();
      names.Add("Sonoo");
      names.Add("Ankit");
      names.Add("Peter");
      names.Add("Irfan");
      names.Add("Ankit");//will not be added

      // Iterate SortedSet elements using foreach loop
      foreach (var name in names)
      {
         Console.WriteLine(name);
      }
```

Stack<T>

FortuneCloud
We Understand Technology

- Stack<T> class is used to push and pop elements. It uses the concept of Stack that arranges elements in LIFO (Last In First Out) order. It can have duplicate elements. It is found in System.Collections.Generic namespace.
- Example:

```
public class StackExample
{
    public static void Main(string[] args)
    {
        Stack<string> names = new Stack<string>();
        names.Push("Sonoo");
        names.Push("Peter");
        names.Push("James");
        names.Push("Ratan");
        names.Push("Irfan");

        foreach (string name in names)
        {
            Console.WriteLine(name);
        }

        Console.WriteLine("Peek element: "+names.Peek());
        Console.WriteLine("Pop: "+ names.Pop());
        Console.WriteLine("After Pop, Peek element: " + names.Peek());
```

Queue<T>

FortuneCloud
We Understand Technology

- Queue<T> class is used to Enqueue and Dequeue elements. It uses the concept of Queue that arranges elements in FIFO (First In First Out) order. It can have duplicate elements. It is found in System.Collections.Generic namespace.
- Example:

```
public class QueueExample
{
    public static void Main(string[] args)
    {
        Queue<string> names = new Queue<string>();
        names.Enqueue("Sonoo");
        names.Enqueue("Peter");
        names.Enqueue("James");
        names.Enqueue("Ratan");
        names.Enqueue("Irfan");

        foreach (string name in names)
        {
            Console.WriteLine(name);
        }

        Console.WriteLine("Peek element: "+names.Peek());
        Console.WriteLine("Dequeue: "+ names.Dequeue());
        Console.WriteLine("After Dequeue, Peek element: " + names.Peek());
```

LinkList<T>

FortuneCloud
We Understand Technology

- LinkedList<T> class uses the concept of linked list. It allows us to insert and delete elements fastly. It can have duplicate elements. It is found in System.Collections.Generic namespace.
- Example:

```
public class LinkedListExample
{
    public static void Main(string[] args)
    {
        // Create a list of strings
        var names = new LinkedList<string>();
        names.AddLast("Sonoo Jaiswal");
        names.AddLast("Ankit");
        names.AddLast("Peter");
        names.AddLast("Irfan");
        names.AddFirst("John");//added to first index

        // Iterate list element using foreach loop
        foreach (var name in names)
        {
            Console.WriteLine(name);
        }
```

Dictionary<Tkey, TValue>

FortuneCloud
We Understand Technology

- Dictionary<TKey, TValue> class uses the concept of hashtable. It stores values on the basis of key. It contains unique keys only. By the help of key, we can easily search or remove elements. It is found in System.Collections.Generic namespace.
- Example:

```
public class DictionaryExample
{
   public static void Main(string[] args)
   {
      Dictionary<string, string> names = new Dictionary<string, string>();
      names.Add("1","Sonoo");
      names.Add("2","Peter");
      names.Add("3","James");
      names.Add("4","Ratan");
      names.Add("5","Irfan");

      foreach (KeyValuePair<string, string> kv in names)
      {
         Console.WriteLine(kv.Key+" "+kv.Value);
      }
```

SortedDictionary<Tkey, TValue>

FortuneCloud
We Understand Technology

- SortedDictionary<TKey, TValue> class uses the concept of hashtable. It stores values on the basis of key. It contains unique keys and maintains ascending order on the basis of key. By the help of key, we can easily search or remove elements. It is found in System.Collections.Generic namespace.
- Example:

```
public class SortedDictionaryExample
{
   public static void Main(string[] args)
   {
      SortedDictionary<string, string> names = new SortedDictionary<string, string>();
      names.Add("1","Sonoo");
      names.Add("4","Peter");
      names.Add("5","James");
      names.Add("3","Ratan");
      names.Add("2","Irfan");
      foreach (KeyValuePair<string, string> kv in names)
      {
         Console.WriteLine(kv.Key+" "+kv.Value);
      }
```

SortedList<Tkey, TValue>

FortuneCloud
We Understand Technology

- SortedList<TKey, TValue> is an array of key/value pairs. It stores values on the basis of key. The SortedList<TKey, TValue> class contains unique keys and maintains ascending order on the basis of key. By the help of key, we can easily search or remove elements. It is found in System.Collections.Generic namespace.
- Example:

```
class Program
   {
      static void Main(string[] args)
      {
         SortedList<string,string>list = new SortedList<String,string>();
         list.Add("1", "One");
         list.Add("2", "Two");
         list.Add("3", "Three");
         list.Add("4", "Four");
         list.Add("5", "Five");
         list.Add("6", "Six");
         list.Add("7", "Seven");
         list.Add("8", "Eight");
         Console.WriteLine("Key and Value of SortedList....");
         foreach (KeyValuePair<string,string> k in list)
{
            Console.WriteLine("Key: {0}, Value: {1}", k.Key, k.Value);
         Console.WriteLine("Is the SortedList having the value? " + list.ContainsValue("Three"));

         Console.WriteLine("Does the SortedList object contains key 10? = " + list.ContainsKey("10"));
}
```

Generic

FortuneCloud
We Understand Technology

- Generic means the general form, not specific. Generic means not specific to a particular data type. It allows you to define generic classes, interfaces, abstract classes, fields, methods, static methods, properties, events, delegates, and operators using the type parameter and without the specific data type. A type parameter is a placeholder for a particular type specified when creating an instance of the generic type.
- A generic type is declared by specifying a type parameter in an angle brackets after a type name, e.g. TypeName<T> where T is a type parameter.

Example

FortuneCloud

```
using System;
namespace CSharpProgram
{
    class GenericClass<T>
    {
        public GenericClass(T msg)
        {
            Console.WriteLine(msg);
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            GenericClass<string> gen   = new GenericClass<string> ("This is generic class");
            GenericClass<int>    genI  = new GenericClass<int>(101);
            GenericClass<char>   getCh = new GenericClass<char>('I');
        }
```

Delegates

FortuneCloud
We Understand Technology

- Delegate is a *reference to the method*. It works like *function pointer* in C and C++. But it is objected-oriented, secured and type-safe than function pointer.
- For static method, delegate encapsulates method only. But for instance method, it encapsulates method and instance both.
- The best use of delegate is to use as event.
- Internally a delegate declaration defines a class which is the derived class of **System.Delegate**.

Example

FortuneCloud

```
using System;
delegate int Calculator(int n);//declaring delegate

public class DelegateExample
{
   static int number = 100;
   public static int add(int n)
   {
      number = number + n;
      return number;
   }
   public static int mul(int n)
   {
      number = number * n;
      return number;
   }
   public static int getNumber()
   {
      return number;
   }
   public static void Main(string[] args)
   {
      Calculator c1 = new Calculator(add);//instantiating delegate
      Calculator c2 = new Calculator(mul);
      c1(20);//calling method using delegate
      Console.WriteLine("After c1 delegate, Number is: " + getNumber());
      c2(3);
      Console.WriteLine("After c2 delegate, Number is: " + getNumber());

   }
```

Reflection

FortuneCloud
We Understand Technology

- Reflection is a *process to get metadata of a type at runtime*. The System.Reflection namespace contains required classes for reflection such as:

1. Type
2. MemberInfo
3. ConstructorInfo
4. MethodInfo
5. FieldInfo
6. PropertyInfo
7. TypeInfo
8. EventInfo
9. Module
10. Assembly
11. AssemblyName
12. Pointer etc.

1.Type Class

FortuneCloud
We Understand Technology

- Type class represents type declarations for class types, interface types, enumeration types, array types, value types etc. It is found in System namespace.
- It inherits System.Reflection.MemberInfo class.
- Example:

```
using System;
public class ReflectionExample
{
  public static void Main()
  {
     int a = 10;
     Type type = a.GetType();
     Console.WriteLine(type);
  }
}
```

2.Get Assembly

FortuneCloud
We Understand Technology

```
using System;
using System.Reflection;
public class ReflectionExample
{
   public static void Main()
    {
        Type t = typeof(System.String);
        Console.WriteLine(t.Assembly);
   }
}
```

3. Print Type Information

```
using System;
using System.Reflection;
public class ReflectionExample
{
   public static void Main()
   {
      Type t = typeof(System.String);
      Console.WriteLine(t.FullName);
      Console.WriteLine(t.BaseType);
      Console.WriteLine(t.IsClass);
      Console.WriteLine(t.IsEnum);
      Console.WriteLine(t.IsInterface);
   }
}
```

4.Print Constructor

FortuneCloud
We Understand Technology

```
using System;
using System.Reflection;
public class ReflectionExample
{
   public static void Main()
   {
      Type t = typeof(System.String);

      Console.WriteLine("Constructors of {0} type...", t);
      ConstructorInfo[] ci = t.GetConstructors(BindingFlags.Public | BindingFlags.Instance);
      foreach (ConstructorInfo c in ci)
      {
         Console.WriteLine(c);
      }
   }
```

5.Print Methods

FortuneCloud
We Understand Technology

```
using System.Reflection;
public class ReflectionExample
{
   public static void Main()
   {
      Type t = typeof(System.String);

      Console.WriteLine("Methods of {0} type...", t);
      MethodInfo[] ci = t.GetMethods(BindingFlags.Public | BindingFlags.Instance);
      foreach (MethodInfo m in ci)
      {
         Console.WriteLine(m);
      }
```

6.Print Fields

FortuneCloud
We Understand Technology

```
using System.Reflection;
public class ReflectionExample
{
    public static void Main()
    {
        Type t = typeof(System.String);

        Console.WriteLine("Fields of {0} type...", t);
        FieldInfo[] ci = t.GetFields(BindingFlags.Public | BindingFlags.Static | BindingFlags.NonPublic);
        foreach (FieldInfo f in ci)
        {
            Console.WriteLine(f);
        }
```

Lambda Expression

FortuneCloud
We Understand Technology

- Lambda expressions are anonymous functions that contain expressions or sequence of operators. All lambda expressions use the lambda operator =>, that can be read as "goes to" or "becomes". The left side of the lambda operator specifies the input parameters and the right side holds an expression or a code block that works with the entry parameters. Usually lambda expressions are used as predicates or instead of delegates (a type that references a method).
- **Expression Lambdas**

Parameter => expression
Parameter-list => expression
Count => count + 2;
Sum => sum + 2;
n => n % 2 == 0

Example

FortuneCloud
We Understand Technology

```
using System;
using System.Collections.Generic;
using System.Linq;
public static class demo
{
   public static void Main()
   {
      List<int> list = new List<int>() { 1, 2, 3, 4, 5, 6 };
      List<int> evenNumbers = list.FindAll(x => (x % 2) == 0);

      foreach (var num in evenNumbers)
      {
         Console.Write("{0} ", num);
      }
      Console.WriteLine();
      Console.Read();

   }
}
```

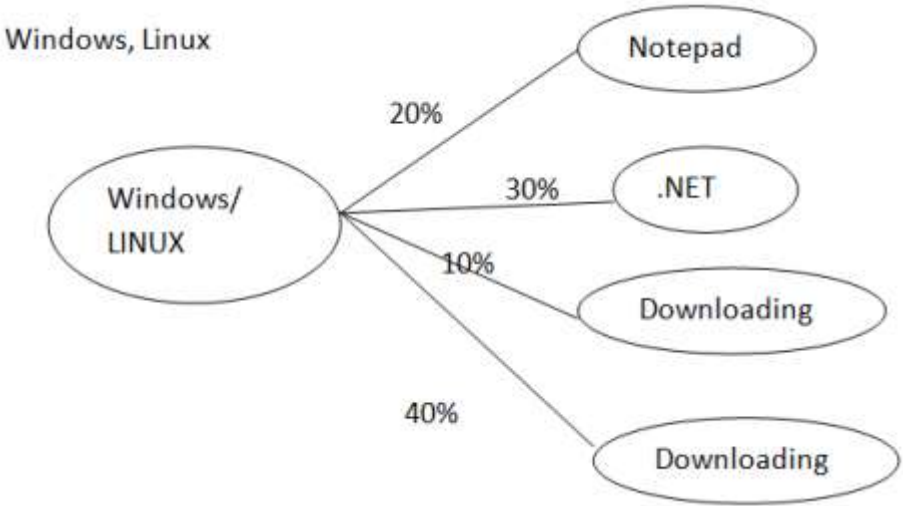
Multithreading

FortuneCloud
We Understand Technology

- Multithreading is a process in which multiple threads works simultaneously. It is a process to achieve multitasking. It saves time because multiple tasks are being executed at a time. To create multithreaded application in C#, we need to use System.Threding namespace.
- **Process and Thread:**

A process represents an application whereas a thread represents a module of the application. Process is heavyweight component whereas thread is lightweight. A thread can be termed as lightweight Subprocess because it is executed inside a process.
Whenever you create a process, a separate memory area is occupied. But threads share a common memory area.

Real-time Example



Example

FortuneCloud
We Understand Technology

```
using System.Threading;
public class Example
{
   public static void thrd1()
   {
        Console.WriteLine("Hello World!!");
    }
    public static void thrd2()
    {
        Console.WriteLine("Today is a great day!!");
     }
}
public class thrd
{
   public static void Main()
   {
       Thread x = new Thread(new ThreadStart(Example.thrd1));
       Thread y = new Thread(new ThreadStart(Example.thrd2));
       x.Start();
       y.Start();
}
```

Thread Life Cycle

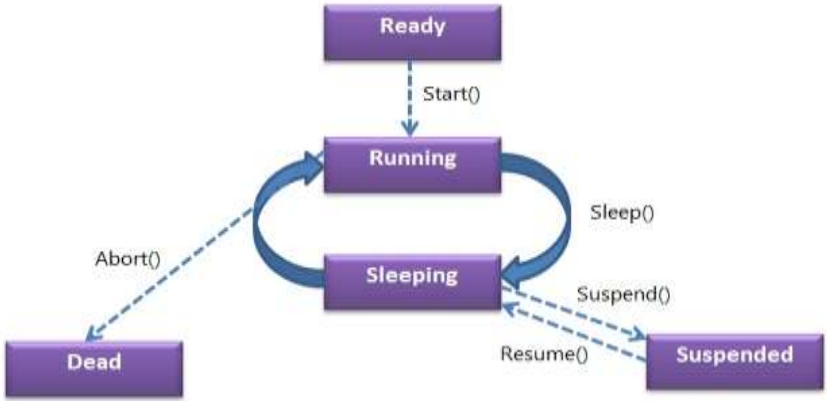


Fig: Thread Life Cycle in C#

Thread Class

FortuneCloud
We Understand Technology

- Thread class provides properties and methods to create and control threads. It is found in System.Threading namespace.

Thread Properties

FortuneCloud
We Understand Technology

Property	Description
CurrentThread	returns the instance of currently running thread.
IsAlive	checks whether the current thread is alive or not. It is used to find the execution status of the thread.
IsBackground	is used to get or set value whether current thread is in background or not.
ManagedThreadId	is used to get unique id for the current managed thread.
Name	is used to get or set the name of the current thread.
Priority	is used to get or set the priority of the current thread.
ThreadState	is used to return a value representing the thread state.

Thread Methods

FortuneCloud
We Understand Technology

Method	Description
Abort()	is used to terminate the thread. It raises ThreadAbortException.
Interrupt()	is used to interrupt a thread which is in *WaitSleepJoin* state.
Join()	is used to block all the calling threads until this thread terminates.
ResetAbort()	is used to cancel the Abort request for the current thread.
Resume()	is used to resume the suspended thread. It is obselete.
Sleep(Int32)	is used to suspend the current thread for the specified milliseconds.
Start()	changes the current state of the thread to Runnable.
Suspend()	suspends the current thread if it is not suspended. It is obselete.
Yield()	is used to yield the execution of current thread to another thread.

Main Thread

FortuneCloud
We Understand Technology

- The first thread which is created inside a process is called Main thread. It starts first and ends at last.
- Example:

```
using System.Threading;
public class ThreadExample
{
   public static void Main(string[] args)
   {
      Thread t = Thread.CurrentThread;
      t.Name = "MainThread";
      Console.WriteLine(t.Name);
   }
}
```

Threading Example: With Static Method

FortuneCloud
We Understand Technology

```
using System.Threading;
public class MyThread
{
    public static void Thread1()
    {
        for (int i = 0; i < 10; i++)
        {
            Console.WriteLine(i);
        }
    }
}
public class ThreadExample
{
    public static void Main()
    {
        Thread t1 = new Thread(new ThreadStart(MyThread.Thread1));
        Thread t2 = new Thread(new ThreadStart(MyThread.Thread1));
        t1.Start();
        t2.Start();
    }
```

Threading Example: With Non-Static Method

FortuneCloud
We Understand Technology

```
using System.Threading;
public class MyThread
{
   public void Thread1()
   {
      for (int i = 0; i < 10; i++)
      {
         Console.WriteLine(i);
      }
   }
}
public class ThreadExample
{
   public static void Main()
   {
      MyThread mt = new MyThread();
      Thread t1 = new Thread(new ThreadStart(mt.Thread1));
      Thread t2 = new Thread(new ThreadStart(mt.Thread1));
      t1.Start();
      t2.Start();
   }
```

Threading Example: With different task on each method

FortuneCloud
We Understand Technology

```
using System.Threading;

public class MyThread
{
   public static void Thread1()
   {
      Console.WriteLine("task one");
   }
   public static void Thread2()
   {
      Console.WriteLine("task two");
   }
}
public class ThreadExample
{
   public static void Main()
   {
      Thread t1 = new Thread(new ThreadStart(MyThread.Thread1));
      Thread t2 = new Thread(new ThreadStart(MyThread.Thread2));
      t1.Start();
      t2.Start();
   }
```

Thread Sleep()

FortuneCloud
We Understand Technology

```
using System.Threading;
public class MyThread
{
   public void Thread1()
   {
      for (int i = 0; i < 10; i++)
      {
         Console.WriteLine(i);
         Thread.Sleep(200);
      }
   }
}
public class ThreadExample
{
   public static void Main()
   {
      MyThread mt = new MyThread();
      Thread t1 = new Thread(new ThreadStart(mt.Thread1));
      Thread t2 = new Thread(new ThreadStart(mt.Thread1));
      t1.Start();
      t2.Start();
   }
```

Thread Abort()

FortuneCloud
We Understand Technology

```
public class MyThread
{
    public void Thread1()
    {
        for (int i = 0; i < 10; i++)
        {
            Console.WriteLine(i);
            Thread.Sleep(200);
        }
    }
}
public class ThreadExample
{
    public static void Main()
    {
        Console.WriteLine("Start of Main");
        MyThread mt = new MyThread();
        Thread t1 = new Thread(new ThreadStart(mt.Thread1));
        Thread t2 = new Thread(new ThreadStart(mt.Thread1));

        t1.Start();
        t2.Start();
        try
        {
            t1.Abort();
            t2.Abort();
        }
        catch (ThreadAbortException tae)
        {
            Console.WriteLine(tae.ToString());
        }
        Console.WriteLine("End of Main");
    }
```

Thread Join()

FortuneCloud
We Understand Technology

```
public class MyThread
{
    public void Thread1()
    {
        for (int i = 0; i < 5; i++)
        {
            Console.WriteLine(i);
            Thread.Sleep(200);
        }
    }
}
public class ThreadExample
{
    public static void Main()
    {
        MyThread mt = new MyThread();
        Thread t1 = new Thread(new ThreadStart(mt.Thread1));
        Thread t2 = new Thread(new ThreadStart(mt.Thread1));
        Thread t3 = new Thread(new ThreadStart(mt.Thread1));
        t1.Start();
        t1.Join();
        t2.Start();
        t3.Start();
```

Thread Naming

FortuneCloud
We Understand Technology

```
public class MyThread
{
    public void Thread1()
    {
        Thread t = Thread.CurrentThread;
        Console.WriteLine(t.Name+" is running");
    }
}
public class ThreadExample
{
    public static void Main()
    {
        MyThread mt = new MyThread();
        Thread t1 = new Thread(new ThreadStart(mt.Thread1));
        Thread t2 = new Thread(new ThreadStart(mt.Thread1));
        Thread t3 = new Thread(new ThreadStart(mt.Thread1));
        t1.Name = "Player1";
        t2.Name = "Player2";
        t3.Name = "Player3";
        t1.Start();
        t2.Start();
        t3.Start();
```

Thread Priority

FortuneCloud
We Understand Technology

```
public class MyThread
{
    public void Thread1()
    {
        Thread t = Thread.CurrentThread;
        Console.WriteLine(t.Name+" is running");
    }
}
public class ThreadExample
{
    public static void Main()
    {
        MyThread mt = new MyThread();
        Thread t1 = new Thread(new ThreadStart(mt.Thread1));
        Thread t2 = new Thread(new ThreadStart(mt.Thread1));
        Thread t3 = new Thread(new ThreadStart(mt.Thread1));
        t1.Name = "Player1";
        t2.Name = "Player2";
        t3.Name = "Player3";
        t3.Priority = ThreadPriority.Highest;
        t2.Priority = ThreadPriority.Normal;
        t1.Priority = ThreadPriority.Lowest;

        t1.Start();
        t2.Start();
        t3.Start();
```

FortuneCloud

We Understand Technology