Abstract Classes and Abstract Methods

A method without any method body is known as an abstract method. What the method contains is only declaration of the method.

A class under which we define abstract methods is known as an abstract class.

Note: To define a method or class as abstract we require to use the abstract keyword.

```
Abstract class Math
{
        Public abstract void Add(int x,int y);
}
```

If a method is declared as abstract under any class the child class of that is responsible for implementing the method without fail.

The concept of abstract methods will be nearly similar to the concept of Method Overriding.

In Overriding Method

```
class Class1
{
        public virtual void show()
        {
        }
}

class Class2:Class1
{
        public override void show()// Optional
        {
                Re-Implementation
        }
}
```

In Abstract Method

```
abstract class Class1
{
        public abstract void show()
}

class Class2:Class1
{
        public override void show()// Mandatory
        {
                Implementation
        }
}
```

The concept of abstract method is similar to method overriding.
Abstract Class: Abstract class can contain abstract methods as well as non abstract methods.

Child class of Abstract Class:

Child class has to implement each and every abstract methods of parent class.
Now only we can consume non-abstract methods of parent class.
It is the combination of assets and liability.

First program on abstract

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace abstractproj
{
	class Abschild:Absparent
	{
		public override void Mul(int x, int y)
		{
			Console.WriteLine(x * y);
		}
		public override void Div(int x, int y)
		{
			Console.WriteLine(x / y);
		}
		static void Main()
		{
			Abschild c = new Abschild();
			Absparent p = c;  //Reference of a parent class by using a child class instance
			c.Add(150, 45);
			c.Sub(78, 23);
			c.Mul(45, 5);
			c.Div(45, 5);
			Console.ReadLine();
		}
	}
}
```

When logic vary from class to class in the child in such scenarios we cannot define the method in the parent, we simply declared in the parent as abstract and make the child classes to implement the method.

Second program on abstract

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace AbstractImplementation
{
	public abstract class Figure
	{
		public double Width, Height, Radius;
		public double Pi = 3.14;
		public abstract double GetArea();
	}
	public class Rectangle : Figure
	{
		public Rectangle(double Width, double Height)
		{
			this.Width = Width;
			this.Height = Height;
		}
		public override double GetArea()
		{
			return Width * Height;
		}
	}
	public class Circle : Figure
	{
		public Circle(double Radius)
		{
			this.Radius = Radius;
		}
		public override double GetArea()
		{
			return (Pi * Radius * Radius);
		}
	}
	public class Cone : Figure
	{
		public Cone(double Radius, double Height)
		{
			this.Radius = Radius;
			this.Height = Height;
		}
		public override double GetArea()
		{
			return Pi * Radius * (Radius + Math.Sqrt(Height * Height));
		}
	}
```

```
        class TestFigure
        {
                static void Main()
                {
                        Rectangle r = new Rectangle(12.67, 56.78);
                        Circle c = new Circle(45.67);
                        Cone cone = new Cone(34.98, 12.98);
                        Console.WriteLine("Area of Rectangle : "+ r.GetArea());
                        Console.WriteLine("Area of Circle : "+ c.GetArea());
                        Console.WriteLine("Area of Cone: "+ cone.GetArea());
                        Console.ReadLine();
                } //End of Main
        } //End of class testfigure
} // End of namespace
```

*****

Interface and Inheritance-based Polymorphism

Class: It’s a user-defined data type with method.
Class: Non-Abstract Methods (Method with Method body)
Abstract Class: A Method without Method body, known as Abstract Class.
Interface: This is also a user defined data type only. Contains only Abstract Methods (Methods without Method body).
Note: Every abstract method of an interface should be implemented by the child class of the interface without fail (mandatory).
- Generally a class inherits from another class to consume the members of its parent, where as if a class is inheriting from an interface it is to implement the members of its parent.
- A class can inherit from a class and interface at a time.

Now, in Visual Studio, we create a new project Add New item in the project and in this we find an item name **Interface** and give a name for it. (like testinterface)
Syntax of interface declaration:

```
[modifiers] interface <name>
{
        -   Abstract member declaration here
}
```

public abstract void Add(int a, int b); // this declaration is for class.
Void Add(int a, int b); // this declaration for interface.

By default scope for the member of an interface is public.

By default every member of an interface is abstract so we don’t require using abstract modifier on it again just like we do in case of class.

- **We can’t declare any fields/variables under an interface.**

```
namespace interfaceproject
{
        Interface testinterface
        {
                Int x;  // error interface can’t contains fields/variables.
                void Add(int a, int b);
                void Sub(int a, int b);
        }
 }
```

- **If required interface can inherit from another interface.**

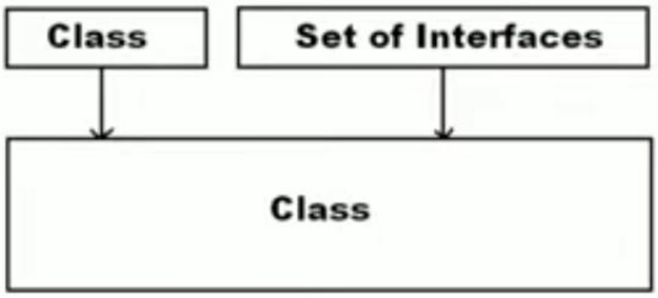
```
namespace interfaceproject
{
        Interface testinterface1
        {
                Int x;  // error interface can’t contains fields/variables.
                void Add(int a, int b);
        }
        Interface testinterface2 : testinterface1                    // Two separate interface
        {
                void Sub(int a, int b);
        }
 }
```

- **Every member of an interface should be implemented under the child class of the interface without fail, but while implementing we don't require to use override modifier just like we have done in case of abstract class.**

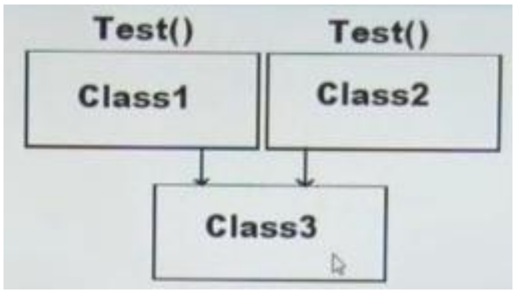
```
namespace interfaceproject
{
        Interface testinterface1
        {
                Int x;  // error interface can't contains fields/variables.
                void Add(int a, int b);
        }
        Interface testinterface2 : testinterface1                    // Two separate interface
        {
                void Sub(int a, int b);
        }
        //Class ImplementationClass : testinterface1 //error ImplementationClass does not implement
                                                      // interface member
        Class ImplementationClass : testinterface2
        {
                public void Add(int a, int b)
                {
                        Console.WriteLine(a + b);
                }
                void testinterface1.Sub(int a, int b) // another way of implementation.
                {
                        Console.WriteLine(a – b);
                }
        /*      Static void Main()
                {
                        ImplementationClass obj = new ImplementationClass();
                        Obj.Add(90,30);
                        Sub(77,66);
                        Console.ReadLine();
                }*/
                Static void Main()
                {
                        ImplementationClass obj = new ImplementationClass();
                        Testinterface2 xyz = obj;  // Reference of the interface, here xyz is a reference
                                                   // created by using a the child instance.
                        xyz.Add(90,30);
                        xyz.Sub(77,66);
                        Console.ReadLine();
                }
        }
}
```

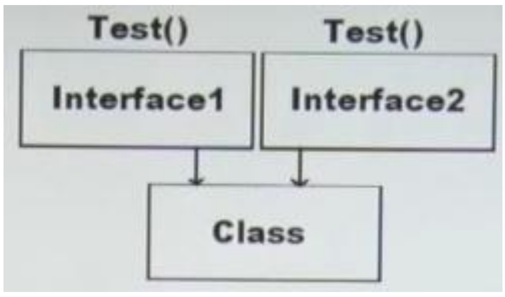
Multiple inheritance with Interface

- Even if multiple inheritance is not supported thru classes in CSharp, it is sill supported thru interface
- A class can have one and only one immediate parent class, whereas the same class can have any number of interface as it's parent i.e. multiple inheritance is supported in CSharp thru interfaces.



Q. Why multiple inheritances are not supported thru classes and how is it supported thru interfaces?
Ans. Multiple inheritance is supported thru classes because Ambiguity problem





Multiple inheritance with Interface

Example

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace interfaceproj
{
        interface Interface1
        {
                void Test();
                void show();
        }
        interface Interface2
        {
                void Test();
                void show();
        }
        class MultipleInheritanceTest:Interface1, Interface2
        {
                public void Test()
                {
                        Console.WriteLine("Interfaces method implemented in child class");
                }

                void Interface1.show()
                {
                        Console.WriteLine("Declared in interface1 and implement in class");
                }
                void Interface2.show()
                {
                        Console.WriteLine("Declared in interface2 and implement in class");
                }

                static void Main()
                {
                        MultipleInheritanceTest abc=new MultipleInheritanceTest();
                        abc.Test();
                        Interface1 s1= abc;
                        Interface2 s2= abc;
                        s1.show();
                        s2.show();
                        Console.ReadLine();
                }
        }
}
```

This Example Implementation of interfaces and the use of polymorphism.

```
// Using Interfaces and Polymorphism

using System;

// interface declaration
interface Vehicle
{
     // abstract method.
    void speedUp(int a);
}

// class implements interface
class Bicycle : Vehicle
{
    int speed;

    // Method increase speed
    public void speedUp(int increment)
    {
        speed = speed + increment;
    }

    // Method check speed
    public void CheckSpeed()
    {
        Console.WriteLine("speed: " + speed);
    }
}

// class implements interface
class Bike : Vehicle
{
    int speed;

    // to increase speed
    public void speedUp(int increment)
    {
        speed = speed + increment;
    }
    public void CheckSpeed()
    {
        Console.WriteLine("speed: " + speed);
    }
}
```

```
class Geeks
{
        public static void Main(String[] args)
        {
                // creating an instance of Bicycle
                // doing some operations
                Bicycle bicycle = new Bicycle();
                bicycle.speedUp(3);

                Console.WriteLine("Bicycle present state :");
                bicycle.CheckSpeed();

                // creating instance of bike.
                Bike bike = new Bike();
                bike.speedUp(4);

                Console.WriteLine("Bike present state :");
                bike.CheckSpeed();
        }
}
```

Output

```
Bicycle present state :
speed: 3
Bike present state :
speed: 4
```

Advantages of Interfaces

Loose coupling: It is used to achieve loose coupling rather than concrete implementations, we can decouple classes and reduce interdependencies.
Abstraction: Interface helps to achieve full abstraction.
Maintainability: It helps to achieve component-based programming that can make code more maintainable.
Multiple Inheritance: Interface used to achieve multiple inheritance and abstraction.
Define architecture: Interfaces add a plug and play like architecture into applications.
Modularity: Promote a cleaner and more modular design. By separating the definition of behavior from the implementation

*****

Delegates

Delegates are similar to interface, we can say that delegates are similar to function pointers in C++.
CSharp delegates are much safer than function pointers, due to their **type-safety**.
Delegates is of reference type
Delegates signature should be as same as the method signature referencing by a delegates
Delegates can point to a parameterized method or non parameterized method.
Delegates has no body.
We can use invoke() method with delegates.
It is used to encapsulate methods.
Delegates are objects that contain reference to method that need to be invoke instead of containing the actual method names.
In Csharp, invoking a delegate will execute the referenced method at RUN TIME.

There are three steps in defining and using delegates:

1. Declaration
2. Instantiation
3. Invocation

Example

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Delegateproj
{
         //All functions return type and parameters no. as well as type must be same.
        public delegate void Calculation(int a, int b);   // Delegate Declaration
        class Program1
        {
                public static void Addition(int a, int b)
                {
                        int result = a + b;
                        Console.WriteLine("Addition result is  {0}", result);
                }

                public static void Substraction(int a, int b)
                {
                        int result = a - b;
                        Console.WriteLine("Substraction result is  {0}", result);
                }

                public static void Multiplication(int a, int b)
                {
                        int result = a * b;
                        Console.WriteLine("Multiplication result is  {0}", result);
                }

                public static void Division(int a, int b)
                {
                        int result = a / b;
                        Console.WriteLine("Division result is  {0}", result);
                }
```

```
                    static void Main(string[] args)
                    {
                              Calculation obj7 = new Calculation(Program1.Addition);  // Instantiation a delegate
                              obj7.Invoke(20, 10);
                              obj7 = Substraction;
                              obj7.Invoke(20, 10);  // Invoking a delegate
                              obj7 = Multiplication;
                              obj7(20, 10); // Invoking a delegate
                              obj7 = Division;
                              obj7(20, 10); // Invoking a delegate
                              Console.ReadLine();
                    }
          }
}
```

Multi Cast Delegates

In Multi Cast Delegate, Delegate can hold the reference of more than one method, can be called with the help of Delegate.

Example

//First Program without Delegate

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Delegateproj2
{
          Class Rectangle
          {
                    public void GetArea(double Width, double Height)
                    {
                              Console.WriteLine(Width * Height);
                    }

                    public void GetPerimeter(double Width, double Height)
                    {
                              Console.WriteLine(2 * (Width * Height);
                    }

                    Static void Main()
                    {
                              Rectangle Rect = new Rectangle();
                              Rect.GetArea(12.34 , 76.34);
                              Rect.GetPerimeter(12.34 , 76.34);

                              Console.ReadLine();
                    }
          }
}
```

//Second Program with Delegate

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Delegateproj2
{
        Public delegate void Rectdelegate(double Width , double Height)
        Class Rectangle
        {
                public void GetArea(double Width, double Height)
                {
                        Console.WriteLine(Width * Height);
                }

                public void GetPerimeter(double Width, double Height)
                {
                        Console.WriteLine(2 * (Width * Height);
                }

                Static void Main()
                {
                        Rectangle Rect = new Rectangle();
                        //RectDelegate obj = new RectDelegate(Rect.GetArea); //or
                        Rectdelegate obj = Rect.GetArea; // Both are equal

                        Obj += Rect.GetPerimeter; // Here, we binding to methods with Delegate. It is
                                                  // called Multi Cast Delegate

                        obj(12.37 , 23,21); //or
                        //obj.Invoke(12.37,23.21);//Either obj(12.37,23,21); or obj.Invoke(12.37, 23.21);

                        Console.ReadLine();
                }
        }
}
```

Note : When you need to perform action on same values you can use Multi Cast Delegate.

Anonymous Methods

In **Anonymous Methods**, without binding a named method to the delegate, you can bind the Code Block to the Delegate.

Example

Program - 1

```
// Non – Anonymous Method
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Delegateproj3
{
        public delegate string GreetingDelegate(string name);
        class AnonymousMethods
        {
                public static string Greetings(string name)
                {
                        return(“Hello ”+name+” a very good morning”);
                }

                static void Main()
                {
                        GreetingDelegate obj = new GreetingDelegate(Greetings);

                        string str = obj.Invoke(“Mohan”);
                        Console.WriteLIne(str);
                        Console.ReadLine();
                }
        }
}
```

Example

Program - 2

// Now, Convert above program in to Anonymous Method program.

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Delegateproj3
{
        public delegate string GreetingDelegate(string name);
        class AnonymousMethods
        {
                static void Main()
                {
                        GreetingDelegate obj = delegate(string name) // Anonymous Method (Logics
                                        //are implemented at the time of Binding has been performed
                        {
                                return(“Hello ”+name+” a very good morning”);
                        };
                        string str = obj.Invoke(“Mohan”);
                        Console.WriteLIne(str);
                        Console.ReadLine();
                }
        }
}
```

Notes: Anonymous Method is not suggested every time. Anonymous Method is suggested when codes of method are lesser (20 lines or 30 lines).

Predefined Generic delegates

There are three predefine delegates available for us.

1. **Func –** Func delegate is used when your method is going to return a value.
2. **Action -** Func delegate is used when your method is going to void. (return nothing)
3. **Predicate -** Func delegate is used when your method is going to return a boolean value.

Without using predefined delegates

```
using System
Namespace DelegatesProject
{
        public delegate double Delegate1(int x, float y, double z);
        public delegate void  Delegate2(int x, float y, double z);
        public delegate bool Delegate3(string str);
        class GenericDelegates
        {
                public static double AddNums1(int x, float y, double z)
                {
                        return x+y+z;
                }

                public static void AddNums2(int x, float y, double z)
                {
                        Console.WriteLine(x+y+z);
                }

                public static bool CheckLength(string str)
                {
                        If (str.Length>5)
                        {
                                return true;
                        }
                        else
                        {
                                return false;
                        }
                }
                static void Main()
                {
                        Delegate1 obj1 = AddNum1;
                        Double result = obj1(100, 34.5f, 193.432);
                        Console.WriteLine(result);

                        Delegate2 obj2 = AddNum2;
                        obj2(100, 34.5f, 193.432);

                        Delegate3.obj3=CheckLength;
                        bool status = Obj3("Hello ! World");
                        Console.WriteLine(status);
                        Console.ReadLine();
                }
        }
}
```

Using predefined delegates

```
using System
Namespace DelegatesProject
{
        class GenericDelegates
        {
                public static double AddNums1(int x, float y, double z)
                {
                        return x+y+z;
                }

                public static void AddNums2(int x, float y, double z)
                {
                        Console.WriteLine(x+y+z);
                }

                public static bool CheckLength(string str)
                {
                        If (str.Length>5)
                        {
                                return true;
                        }
                        else
                        {
                                return false;
                        }
                }
                static void Main()
                {
                        Func<int, float, double, double) obj1 = AddNum1; // Last double is return type
                        Double result = obj1(100, 34.5f, 193.432);
                        Console.WriteLine(result);

                        Action<int, float, double) obj2 = AddNum2;
                        obj2(100, 34.5f, 193.432);

                        Predicate<string> obj3=CheckLength;
                        bool status = Obj3("Hello ! World");
                        Console.WriteLine(status);
                        Console.ReadLine();
                }
        }
}
```

Note: In Func delegate, We can pass up to 16 parameters as input and 1 parameter as return.
In Action delegate, We can pass up to 16 parameters as input only.
In Predicate delegate, only takes one parameter.

Lambda Expressions

Lambda Expressions is the shorthand for writing Anonymous method. Lambda method is used to simplify the concepts of the Anonymous Method.

It is a simple Delegate program. (without using anonymous method and without using Lambda Expressions)

```
Using System;
Using System.Collections.Generic;
Using System.Linq;
Using System.Text;
Using System.Threading.Tasks;

namespace DelegatesProject
{
        public delegate string GreetingDelegate(string name);

        class LambdaExpressions
        {
                public static string Greetings(string name)
                {
                        return “Hello”+name+”a very good morning”;
                }

                static void Main()
                {
                        GreetingsDelegate obj = new GreetingsDelegate(Greetings);
                        string str = obj(“Raja”);
                        Console.WriteLine(str);
                        Console.readLine();
                }
        }
}
```

Now, the above program converted into anonymous Method.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DelegatesProject
{
        public delegate string GreetingDelegate(string name);
        class LambdaExpressions
        {
                static void Main()
                {
                        GreetingsDelegate obj = delegate(string name) //we put the code in delegate
                        {
                                return “Hello”+name+”a very good morning”;
                        };

                        string str = obj(“Raja”);
                        Console.WriteLine(str);
                        Console.readLine();
                }
        }
}
```

Now, the above Anonymous program using **Lambda Expression**.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DelegatesProject
{
        //public delegate string GreetingDelegate(string name);
        class LambdaExpressions
        {
                static void Main()
                {
                        GreetingsDelegate obj = (name) => // It is lambda Expression, we may use
                                                          //“string” with name
                        {
                                return “Hello”+name+”a very good morning”;
                        };

                        string str = obj(“Raja”);
                        Console.WriteLine(str);
                        Console.readLine();
                }
        }
}
```

*****

Multithreading

Thread is a unit which executes the code under an application. It is a lightweight process. Every application by default contains one Thread to execute the program and that is known as Main Thread, so every program is by default single threaded model.
To use Thread, we use:
using System.Threading

We are going to write a program which will execute sequential one by one and we will see what is drawback of the program.

```
using System;
using System.Threading;

class ThreadDemo
{
        static void test1()
        {
                for (int I ; i<=100 ; i++)
                {
                        Console.WriteLine("Test1 : "+i);
                }
        }

        static void test2()
        {
                for (int I ; i<=100 ; i++)
                {
                        Console.WriteLine("Test2 : "+i);
                }
        }

        static void test3()
        {
                for (int I ; i<=100 ; i++)
                {
                        Console.WriteLine("Test3 : "+i);
                }
        }
        static void Main()
        {
                test1();
                test2();
                test3();
                Console.ReadLine();
        }
}
```
Run the program and see the output.
It is a single threaded program run all code one after the other (Completing the first action then go to second action, completing the second action then go to the third action). Drawback of sequential execution is suppose a action take more time to complete their action next process (action) will be delayed, next action will be executed when previous action has completed.

We try to understand the problem by a program.

```
using System;
using System.Threading;

class ThreadDemo
{
        static void test1()
        {
                for (int I ; i<=100 ; i++)
                {
                        Console.WriteLine(“Test1 : “+i);
                }
        }

        static void test2()
        {
                for (int I ; i<=100 ; i++)
                {
                        Console.WriteLine(“Test2 : “+i);

                        If (i==50)
                        {
                                Console.WriteLine(“Test2 Thread going to Sleep”);
                                Thread.Sleep(5000); //Wait for 5 seconds
                                Console.WriteLine(“Now, Test2 Thread woke up”);
                        }
                }
        }

        static void test3()
        {
                for (int I ; i<=100 ; i++)
                {
                        Console.WriteLine(“Test3 : “+i);
                }
        }

        static void Main()
        {
                test1();
                test2();
                test3();
                ConsloeConsole.ReadLine();
        }
}
```

In this program Test2 Method wait for 5 seconds, until or unless Test2 Method is not completed Test3 Method is not executed, means due to delay 5 seconds of Test2, Test3 Method also delayed for 5 seconds. It is the drawback of single threaded program.

To overcome the above problem we use the concept of Multi-Threading.
In Multi-Thread execution takes place simultaneously.
Operating system allocate time to each thread to execute, it is a time sharing concepts, time slots are given to each threads by Operating System totally depend on Operating System how much time it allot. Through the Multi-Threading concepts we optimize the use of CPU resources.

```
using System;
using System.Threading;

class ThreadTest
{
        static void test1()
        {
                for (int I ; i<=100 ; i++)
                {
                        Console.WriteLine(“Test1 : “+i);
                }
        }

        static void test2()
        {
                for (int I ; i<=100 ; i++)
                {
                        Console.WriteLine(“Test2 : “+i);

                        If (i==50)
                        {
                                Console.WriteLine(“Thread2 is going to Sleep”);
                                Thread.Sleep(5000); //Wait for 5 seconds
                                Console.WriteLine(“Now, Thread2 woke up”);
                        }
                }
        }

        static void test3()
        {
                for (int I ; i<=100 ; i++)
                {
                        Console.WriteLine(“Test3 : “+i);
                }
        }

        static void Main()
        {
                Thread T1 = new Thread(Test1);
                Thread T2 = new Thread(Test2);
                Thread T3 = new Thread(Test3);
                T1.Start;
                T2.Start;
                T3.Start;
                ConsloeConsole.ReadLine();
        }
}
```

Question is how many thread running in above program. Answer is four thread First thread is Main thread, second thread is T1, third thread is T2, forth thread is T3.

In thread2 when thread2 is in sleep mode execution does not wait, in the mean time thread1 or thread2 continue executing.

Other Method to start Threads

Option -1

```
static void Main()
{
        // Explicit Performance  (ThreadStart is a delegate)
        ThreadStart obj1 = new ThreadStart(Test1) ;  //The method is bond with the delegate
        ThreadStart obj2 = new ThreadStart(Test2) ;  //The method is bond with the delegate
        ThreadStart obj3 = new ThreadStart(Test3) ;  //The method is bond with the delegate

        Thread T1 = new Thread(obj1); //Delegate bond with the thread now
        Thread T2 = new Thread(obj2); //Delegate bond with the thread now
        Thread T3 = new Thread(obj3); //Delegate bond with the thread now
        T1.Start();
        T2.Start();
        T3.Start();
        ConsloeConsole.ReadLine();
}
```

Option -2

```
static void Main()
{
        ThreadStart obj1 = delegate () { Test1(); };  //The method is bond with the delegate keyword
        ThreadStart obj2 = delegate () { Test2(); };  //The method is bond with the delegate keyword
        ThreadStart obj3 = delegate () { Test3(); };  //The method is bond with the delegate keyword

        Thread T1 = new Thread(obj1); //Directly passed Method Name here. (Implicit performance)
        Thread T2 = new Thread(obj2); // Directly passed Method Name here. (Implicit performance)
        Thread T3 = new Thread(obj3); // Directly passed Method Name here. (Implicit performance)
        T1.Start();
        T2.Start();
        T3.Start();
        ConsloeConsole.ReadLine();
}
```

Option -3

```
static void Main()
{
        ThreadStart obj1 = () => Test1();  // It is a Lambda expression with Lambda  Operator ( => )
        ThreadStart obj2 = () => Test2();  // It is a Lambda expression with Lambda  Operator ( => )
        ThreadStart obj3 = () => Test3();  // It is a Lambda expression with Lambda  Operator ( => )

        Thread T1 = new Thread(obj1); //Directly passed Method Name here. (Implicit performance)
        Thread T2 = new Thread(obj2); // Directly passed Method Name here. (Implicit performance)
        Thread T3 = new Thread(obj3); // Directly passed Method Name here. (Implicit performance)
        T1.Start();
        T2.Start();
        T3.Start();
        ConsloeConsole.ReadLine();
}
```

Parameterized Threading

ThreadStart does not take any parameter. For Parameterized Thread we use Object as a parameter

```
using System;
using System.Threading;

class ThreadTest
{
        static void test1()
        {
                for (int I ; i<=100 ; i++)
                {
                        Console.WriteLine("Test1 : "+i);
                }
        }

        static void test2(object max)   //sending parameter max as an object
        {
                int num = Convert.ToInt32(max);

                for (int I ; i<=num ; i++)
                {
                        Console.WriteLine("Test2 : "+i);
                }
        }

        static void test3()
        {
                for (int I ; i<=100 ; i++)
                {
                        Console.WriteLine("Test3 : "+i);
                }
        }

        static void Main()
        {
                Thread T1 = new Thread(Test1);
                //Parameterized Threading
                ParameterizedThreadStart obj  = new ParameterizedThreadStart(test2);
                Thread T2 = new Thread(obj);
                Thread T3 = new Thread(Test3);

                T1.Start;
                T2.Start(50);
                T3.Start;
                ConsloeConsole.ReadLine();
        }
}
```

Join Operation In Multithreading

```
using System;
using System.Threading;
namespace ThreadJoin
{
        class ThreadJoinTest
        {
                static void test1()
                {
                        Console.WriteLine(“Thread-1 is  Starting”);
                        for (int I ; i<=100 ; i++)
                        {
                                Console.WriteLine(“Test1 : “+i);
                        }
                        Console.WriteLine(“Thread-1 is  Exiting”);
                }
                static void test2()
                {
                        int num = Convert.ToInt32(max);

                        Console.WriteLine(“Thread-2 is  Starting”);
                        for (int I ; i<=num ; i++)
                        {
                                Console.WriteLine(“Test2 : “+i);
                        }
                        Console.WriteLine(“Thread-2 is  Exiting”);
                }
                static void test3()
                {
                        Console.WriteLine(“Thread-3 is  Starting”);
                        for (int I ; i<=100 ; i++)
                        {
                                Console.WriteLine(“Test3 : “+i);
                        }
                        Console.WriteLine(“Thread-3 is  Exiting”);
                }
                static void Main()
                {
                        Console.WriteLine(“Main Thread is Starting”);
                        Thread T1 = new Thread(Test1);
                        Thread T2 = new Thread(Test2);
                        Thread T3 = new Thread(Test3);

                        T1.Start;
                        T2.Start();
                        T3.Start;
                        Console.WriteLine(“Main Thread is Exiting”);
                        ConsloeConsole.ReadLine();
                }
        }
}
```

Run the above code and see the effect carefully, and give more attention to Main Thread (Main Function).
You will see, Main Thread will not wait until all the thread complete the work. In the mean time Main Thread may be exit. The problem is Main Thread should not be allowed to exit in the middle of program. The execution of program completely started from Main Method and should be end with Main Method.
So, we do not allow to Main Thread to out of the program in middle of program.
If we want to Main Thread to wait until all the threads are completing their work, we require to call **Join**.
To do this, please make changes in above program, which is given below:

```
static void Main()
{
        Console.WriteLine("Main Thread is Starting");
        Thread T1 = new Thread(Test1);
        Thread T2 = new Thread(Test2);
        Thread T3 = new Thread(Test3);

        T1.Start;
        T2.Start();
        T3.Start;
        T1.Join();  // Add this Join Method
        T2.Join();  // Add this Join Method
        T3.Join();  // Add this Join Method
        Console.WriteLine("Main Thread is Exiting");
        ConsloeConsole.ReadLine();
}
```

Now, The Main Thread cannot exit from the program until all the child thread finishing the job.
Run and see the effect. Main Thread will exiting from program in last.
We can also give the time period to Child Thread to wait by Main thread until Main Thread will be exited.
e.g.> T1.Join(3000); // Add this Join Method

Explain :Main Thread will wait for 3 seconds to exit T1 thread otherwise Main Thread will exit.

Here, Join method is overloaded.

Thread locking

```
using System;
using System. Threading

namespace ThreadLockSample
{
        class ThreadLock
        {
                public void Display()
                {
                        lock(This) //At a time single thread can execute the Method.
                        {
                                Console.WriteLine("CSharp is an");
                                Thread.Sleep(5000);
                                Console.WriteLine("Object Oriented language");
                        }
                }

                Static void Main()
                {
                        ThreadLocksample obj = new ThreadLockSample();
                        Thread T1 = new Thread(obj.Display);
                        Thread T2 = new Thread(obj.Display);
                        Thread T3 = new Thread(obj.Display);
                        T1.Start();
                        T2.Start();
                        T3.Start3;
                        Console.ReadLine();
                }
        }
}
```

We can use **Lock** Method to restrict multiple access to the code, means one thread can execute the method at a time. It is known as Thread Locking.

*****