**LINQ**

Linq provides a standerdised way to query data from various data sources using a common syntax within programming languages like c# and vb

Use : Allows developers to retrieve, manipulate and transform data from various data sources such as databases, collections, XML, and In-Memory objects

**Anonymous methods/Anonymous functions**

Method without any name

Defined using a delegate

Example:

Public delegate void MyDelegate(int num);

Class Program{

Public static void MyMethod(int num){

Num+=10;

Console.WriteLine(num);

}

Psvm(){

MyDelegate obj=new MyDelegate(MyMethod);

Obj.Invoke(5);

}

}

In the above code if we want to use the anonymous function:

Public delegate void MyDelegate(int num);

Class Program{

Psvm(){

MyDelegate obj=delegate(int a){

A+=10;

Console.WriteLine(a);

}

Obj.Invoke(5);

}

}

Anonymous method is stored in an delegate obj

You need not specify the return type.

It is inferred as the return type as given while declaring data type

Anonymous method return type = delegate return type

So, no need to give the return type

Anonymous method is not a static and instance

Used when the code volume is less

Limitations :

1. Cant use goto, break and continue
2. Cant access ref or out parameters

Note :

1. Define anonymous method using delegate
2. Must be assigned to delegate object
3. Can access outer variables or functions
4. Can be passed as a parameter to another method or function

Example of passing anonymous method as paremeter

public static void MyMethod(MyDelegate del,int a){

a+=10;

del.Invoke(a);

}

In main function :

Program.MyMethod(delegate(int b){b+=10;Console.WriteLine(b)},5)

// output : a=5-> a=15->b+1🡪b=25

**Lambda Expressions**

Introduced in c# 3.0

Works like an anonymous method, easier than anonymous method

Public delegate void myDelegate(int num)

Static void Main(){

MyDelegate obj=delegate(int num){num+=10;cw(num);}

// anonymous method

MyDelegate obj=(a)=>{

Num+=10;

Console.WrirteLine(a); }

In lambda expression no need to specify the datatype like the anonymous method and also we don’t use the delegate keyword.

Lambda expression is shorthand of anonymous function

Used in LINQ

Types :

1. Statement lambda

Input=>{statements};

Doesnot return any value, returns based on delegate

1. Expression lambda

Input=> expression

Returns the evaluated value explicitly

Ex:

MyDelegate obj=(a)=>a\*\*a;

Console.WriteLine(Obj.Invoke(3));

Another example:

MyDelegate obj=(a,b)=>a\*b;

Obj.Invoke(2,3);

**Use of LINQ:**

Query language that can query any data source

Serves as a good entity for middle tier. So, will sit between the middle tier and data access layer.

Present in System.Linq namespace

Common query language to query data from any datasource

A diagram of a system

Description automatically generated

Linq provider is some sort of API, implements IQueryProvider and IQueryable interfaces for a specific data source. Provides ability to write LinQ queries against that data source.

Without linq provider, application wont understand the linq queries and those queries cant be executed.

Linq providers:

1. Linq to objects (against inmemory objects like arrays, collections, generics, etc)
2. Linq to SQL ( allows one to one/many to many mapping between SQL server objects and .net classes
3. Linq to datasets ( to query cached data in a dataset)
4. Linq to entities ( looks similar to linq to sql. But this can work with any type of database like Oracle, MySQL, DB2, etc
5. Linq to XML

**Query Syntax**

Main components :

1. Datasource
2. Query
3. Execution environment

Stages:

1. Initialisation

From data in datasource

1. Condition

Where datafilterCondition

1. Selection

Select data

**LINQ – Lambda expression**

List<int> nums= new List<int> {1,2,3,4,5}

Var nums\_new = nums.Select(x=>x\*5);

**Writing LINQ queries**

1. Getting all students
2. Lambda expressions

List<Student> list=Student.GetAllStudents().Where(student => student.Gender=”Male”);

1. Sql like expression

From student in Student.GetAllStudents where student.Gender=”Male” select student;

**LINQ Aggregate functions**

Int[] num={1,2,3,4,5};

1. To find the smallest number of list

Int result = num.Min();

1. To find smallest even number

Int even = num.Where(a=>a%2==0).Min();

1. To find largest number

Int result = num.Max();

1. To find maximum even number

Int evenmax= num.Where(a=>a%2==0).Max();

1. To find sum of all numbers

Int result = num.Sum();

1. To find sum of even numbers

Int sumeven= num.Where(a=>a%2==0).Sum();

1. To find the count

Int count = num.Count();

1. To find avg

Int avg= num.Average();