C# Advanced Concepts:

**LINQ** -> This is reffered to Language Integrated Query which is basically used for the accessibility between stored data(XML, SQL etc.) and the programming language.

There are two types of systax available which are

1.LINQ Query syntax

2.LINQ method syntax

Standard Link Query operators are ->

1.where

2.ofType

3.OrderBy

4.thenBy

5.GroupBy

6.join

7.groupjoin

8.select

9.average

10.sum

1.Where Clause: *Where operator in Linq is used to give a particular condition based on which we will filter the results. It is used brfore the select operator.*

using System;

string[] member = { "Amit", "Rjev", "Ajay" };

var findNames = from name in member where name.Contains('a') select name;

foreach (var findName in findNames)

{

Console.WriteLine(findName);

}

2.OfType: *if there are different types available then we can use OfType to find out the particular type in an iterator, inumerator or list.*

using System;

using System.Linq;

using System.Collections;

class Program{

public static void Main(string[] args)

{

IList list = new ArrayList();

list.Add(1);

list.Add("Shovon");

list.Add('a');

list.Add(2.3);

list.Add(new Student(){StudentID=1,StudentName="Shovon Raul",Age=22});

var strings = from s in list.OfType<string>()

select s;

var integers = from s in list.OfType<int>()

select s;

var doubles = from s in list.OfType<double>()

select s;

var objects = from s in list.OfType<Student>()

select s;

foreach(var e in objects)

{

Console.WriteLine(e.StudentName);

Console.WriteLine(e.Age);

}

Console.WriteLine();

foreach(var x in strings)

{

Console.WriteLine(x);

}

}

}

public class Student{

public int StudentID { get; set; }

public string StudentName { get; set; }

public int Age { get; set; }

}

OrderBy: *It works same as the sql statements to order the elements, OrderByDescending() method is also there for reverse operation*

using System;

using System.Linq;

using System.Collections.Generic;

class Program{

public static void Main(string[] args)

{

List<Student> studentList = new List<Student>() {

new Student() { StudentID = 1, StudentName = "John", Age = 18 } ,

new Student() { StudentID = 2, StudentName = "Steve", Age = 15 } ,

new Student() { StudentID = 3, StudentName = "Bill", Age = 25 } ,

new Student() { StudentID = 4, StudentName = "Ram" , Age = 20 } ,

new Student() { StudentID = 5, StudentName = "Ron" , Age = 19 }

};

var orderByStudentName = from student in studentList

orderby student.StudentName

select student;

var orderByStudentNamedesc = studentList.OrderByDescending(s=>s.StudentName);

foreach(var stu in orderByStudentName)

{

Console.WriteLine(stu.StudentName);

}

}

}

public class Student{

public int StudentID { get; set; }

public string StudentName { get; set; }

public int Age { get; set; }

}

thenBy*: This works after orderBy*

using System;

using System.Linq;

using System.Collections.Generic;

public class Program

{

public static void Main()

{

// Student collection

IList<Student> studentList = new List<Student>() {

new Student() { StudentID = 1, StudentName = "John", Age = 18 } ,

new Student() { StudentID = 2, StudentName = "Steve", Age = 15 } ,

new Student() { StudentID = 3, StudentName = "Bill", Age = 25 } ,

new Student() { StudentID = 4, StudentName = "Ram" , Age = 20 } ,

new Student() { StudentID = 5, StudentName = "Ron" , Age = 19 }

};

var thenByResult = studentList.OrderBy(s => s.StudentName).ThenBy(s => s.Age);

var thenByDescResult = studentList.OrderBy(s => s.StudentName).ThenByDescending(s => s.Age);

Console.WriteLine("ThenBy:");

foreach (var std in thenByResult)

Console.WriteLine("Name: {0}, Age:{1}", std.StudentName, std.Age);

Console.WriteLine("ThenByDescending:");

foreach (var std in thenByDescResult)

Console.WriteLine("Name: {0}, Age:{1}", std.StudentName, std.Age);

}

}

public class Student{

public int StudentID { get; set; }

public string StudentName { get; set; }

public int Age { get; set; }

}

GroupBy: *Used to grouping the elements.*

using System;

using System.Linq;

using System.Collections.Generic;

class Program{

public static void Main(string[] args)

{

List<Student> studentList = new List<Student>() {

new Student() { StudentID = 1, StudentName = "John", Age = 18 } ,

new Student() { StudentID = 2, StudentName = "Steve", Age = 15 } ,

new Student() { StudentID = 3, StudentName = "Bill", Age = 18 } ,

new Student() { StudentID = 4, StudentName = "Ram" , Age = 18 } ,

new Student() { StudentID = 5, StudentName = "Ron" , Age = 19 }

};

var orderByStudentName = studentList.OrderByDescending(s=>s.StudentName);

foreach(var stu in orderByStudentName)

{

Console.WriteLine(stu.StudentName);

}

Console.WriteLine();

var ageGroup = studentList.GroupBy(s=>s.Age);

foreach(var age in ageGroup){

Console.WriteLine("Age Group: {0}", age.Key);

foreach(Student s in age){

Console.WriteLine(s.StudentName);

}

}

}

}

Join: *Join works same as SQL Join clasue taking the five parameters one after another in the syntax which are 1) outer 2) inner 3) outerKeySelector 4) innerKeySelector 5) resultSelector.*

using System;

using System.Linq;

using System.Collections.Generic;

class Program{

public static void Main(string[] args)

{

List<Student> studentList = new List<Student>() {

new Student() { StudentID = 1, StudentName = "John", StandardID =1 },

new Student() { StudentID = 2, StudentName = "Moin", StandardID =1 },

new Student() { StudentID = 3, StudentName = "Bill", StandardID =2 },

new Student() { StudentID = 4, StudentName = "Ram" , StandardID =2 },

new Student() { StudentID = 5, StudentName = "Ron" }

};

List<Standard> standardList = new List<Standard>() {

new Standard(){ StandardID = 1, StandardName="Standard 1"},

new Standard(){ StandardID = 2, StandardName="Standard 2"},

new Standard(){ StandardID = 3, StandardName="Standard 3"}

};

//INNER JOIN

var newTable = studentList.Join(

standardList,

Student => Student.StandardID,

Standard => Standard.StandardID,

(Student, Standard) => new{

StudentName = Student.StudentName,

StandardName = Standard.StandardName

}

);

//printing the new join

foreach(var obj in newTable)

{

Console.WriteLine(obj);

}

}

}

public class Student{

public int StudentID { get; set; }

public int StandardID { get; set; }

public string StudentName { get; set; }

public int Age { get; set; }

}

public class Standard{

public int StandardID { get; set; }

public string StandardName { get; set; }

}

GroupJoin : *Group join use both the feature of groupBy and join to show the result.*

using System;

using System.Linq;

using System.Collections.Generic;

class Program{

public static void Main(string[] args)

{

List<Student> studentList = new List<Student>() {

new Student() { StudentID = 1, StudentName = "John", StandardID =1 },

new Student() { StudentID = 2, StudentName = "Moin", StandardID =1 },

new Student() { StudentID = 3, StudentName = "Bill", StandardID =2 },

new Student() { StudentID = 4, StudentName = "Ram" , StandardID =2 },

new Student() { StudentID = 5, StudentName = "Ron" }

};

List<Standard> standardList = new List<Standard>() {

new Standard(){ StandardID = 1, StandardName="Standard 1"},

new Standard(){ StandardID = 2, StandardName="Standard 2"},

new Standard(){ StandardID = 3, StandardName="Standard 3"}

};

//INNER JOIN

var newTable = studentList.Join(

standardList,

Student => Student.StandardID,

Standard => Standard.StandardID,

(Student, Standard) => new{

StudentName = Student.StudentName,

StandardName = Standard.StandardName

}

);

var newGroupTable = standardList.GroupJoin(

studentList,

standard => standard.StandardID,

student => student.StandardID,

(standard,student) => new{

standardFullName = standard.StandardName,

students = student

}

);

//printing the new join

foreach(var obj in newTable)

{

Console.WriteLine(obj);

}

Console.WriteLine();

foreach(var obj in newGroupTable)

{

Console.WriteLine(obj.standardFullName);

foreach(var student in obj.students)

{

Console.WriteLine(student.StudentName);

}

}

}

}

public class Student{

public int StudentID { get; set; }

public int StandardID { get; set; }

public string StudentName { get; set; }

public int Age { get; set; }

}

public class Standard{

public int StandardID { get; set; }

public string StandardName { get; set; }

}

**SOLID PRINCILES:**

Solid principles are the best approach invented by Robert C. Martin in 1990s. It is used to transform the tightly coupled code with little encapsulation to the loosely coupled code with more encapsulation and removing redundency.

SOLID Principles refers to –

S **->** **Single Responsibility Principle:**

Every class should have a single responsibility and based on that the class would be created. No other instance’s responsibility should be added to that class.

O **->** **Open- Closed Principle:**

A class should be open for extension and close for modification. It means after creating a class when we need to extend the class feature then only we can update the class, and it should be unchanged until any debugging is required.

L **–> Liscov Substitution Principle:**

Objects of the superclass can be replaced with instances of the subclasses without effecting the code.

I **–> Interface Segregation Principle:**

It is best practice to create small interfaces other than one large interface so the interfaces can be created based on the requirement and the code becomes loosely coupled code.

D **–> Dependency Inversion Principle:**

This principle states that High level modules should not depend on low level modules, but both should depend on abstruct classes and abstruct classes should not be dependent on any other classes.