Pointers : Variables which hold address of other variables

Delegates : Function pointers , pointers which point to functions

To use Delegate, first declare it

We will create some methods whose signature is same

Int add (int x, int y);

Int subtract (int x, int y);

Int product (int x, int y);

Int divide (int x, int y);

Delegate int dele (int x, int y);

using System;

class Program

{

static int add (int x, int y)

{

return x + y;

}

static int subtract(int x, int y)

{

return x - y;

}

static int product(int x, int y)

{

return x \* y;

}

static int divide(int x, int y)

{

return x / y;

}

delegate int del(int x, int y);

static void Main()

{

del delobj = new del(add);

Console.WriteLine(delobj(10,2));

}

}

using System;

class Program

{

static int add (int x, int y)

{

return x + y;

}

static int subtract(int x, int y)

{

return x - y;

}

static int product(int x, int y)

{

return x \* y;

}

static int divide(int x, int y)

{

return x / y;

}

delegate int del(int x, int y);

static void Main()

{

del delobj = new del(add);

Console.WriteLine(delobj(10,2));

delobj = new del(subtract);

Console.WriteLine(delobj(10, 2));

delobj = new del(product);

Console.WriteLine(delobj(10,2));

}

}

Delegates

Single Cast Delegate : Its is a delegate which points to a single method at a time

Multi Cast Delegate : Delegate which points to more than one function at a time

It creates chain of methods

using System;

class Program

{

static void add (int x, int y)

{

Console.WriteLine("Result of Addition is " + (x+y));

}

static void subtract(int x, int y)

{

Console.WriteLine("Result of Subtraction is " + (x - y));

}

static void product(int x, int y)

{

Console.WriteLine("Result of Multiplication is " + (x \* y));

}

static void divide(int x, int y)

{

Console.WriteLine("Result of Division is " + (x / y));

}

delegate void del(int x, int y);

static void Main()

{

del delobj = new del(add);

delobj += new del(subtract); // += creates chain of methods

delobj += new del(product);

delobj += new del(divide);

delobj(20, 10);

Console.WriteLine("After mthod add is removed grom the chain"); ;

delobj -= new del(add);

delobj(20, 10);

}

}

// Declare a delegate

delegate void Del(int i, double j);

class MathClass

{

static void Main()

{

MathClass m = new MathClass();

// Delegate instantiation using "MultiplyNumbers"

Del d = m.MultiplyNumbers;

// Invoke the delegate object.

Console.WriteLine("Invoking the delegate using 'MultiplyNumbers':");

for (int i = 1; i <= 5; i++)

{

d(i, 2);

}

// Keep the console window open in debug mode.

Console.WriteLine("Press any key to exit.");

Console.ReadKey();

}

// Declare the associated method.

void MultiplyNumbers(int m, double n)

{

Console.Write(m \* n + " ");

}

}

---------------------------------------------------------  
Uses of Delegates

1. Events
2. Anonymous Methods

What is a Anonymous Methods > A method which does not have a name

{

statements

}

using System;

class Program

{

delegate void del(int x, int y);

static void Main()

{

//del delobj = new del (add);

del delobj = delegate (int x, int y)

{

Console.WriteLine(x + y);

};

delobj(20, 10);

}

}

**Lambda Expressions : it’s a way to write anonymous methods in a short way**

For Lambda expressions, we use Lambda operator

=>

Input => output

Anonymous Method using Lambda Expression

using System;

class Program

{

delegate void del(int x, int y);

static void Main()

{

//del delobj = new del (add);

//del delobj = delegate (int x, int y)

//{

// Console.WriteLine(x + y);

//};

del delobj = (x, y) =>

{

Console.WriteLine(x + y);

};

delobj(20, 10);

}

}

|  |
| --- |
| using System; |
|  |  |
|  | namespace LambdaExpressions |
|  | { |
|  | delegate void PrintHello(); |
|  |  |
|  | class Program |
|  | { |
|  | static void Main(string[] args) |
|  | { |
|  | PrintHello sayHello = () => |
|  | { |
|  | string name = "Alice"; |
|  | Console.WriteLine($"Hello {name}"); |
|  | }; |
|  | sayHello(); |
|  | Console.WriteLine(); |
|  | } |
|  | } |
|  | } |

<https://itnext.io/delegates-anonymous-methods-and-lambda-expressions-5ea4e56bbd05>

## ****Action and Func Delegates****

Action in C# represents a delegate that has void return type and optional parameters.

An Action delegate can take up to 16 input parameters of different types.

using System;

class Program

{

static void Add(int x, int y)

{

Console.WriteLine("Result is " + (x+y));

}

//delegate void del(int x, int y); // Not needed

static void Main()

{

// List<int>

Action<int, int> sum = Add;

sum(20, 10);

}

}

<https://itnext.io/delegates-anonymous-methods-and-lambda-expressions-5ea4e56bbd05>

LINQ

Language Integrated Query: Query Language which can be used to work with any collection which implements either IEnumerable or IQueryable interface

LINQ

using System;

using System.Linq;

class Program

{

static void Main()

{

int[] num = new int[10] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

//for (int i = 0; i < 10;i++)

//{

// Console.WriteLine(num[i]);

//}

// SQl Query

// Select \* from tablename

// LINQ Query

var temp = from x in num

select x;

foreach(var x in temp)

Console.WriteLine(x);

// Even numbers from this Array

//for (int i = 0; i < 10; i++)

//{

// if(num[i]%2==0)

// Console.WriteLine(num[i]);

//}

// Even numbers from this Array by using LINQ

var eventemp = from x in num

where x % 2 == 0

select x;

Console.WriteLine("Even numbers are ");

foreach(var x in eventemp)

Console.WriteLine(x);

// How many values are there in this Array

var count = (from x in num

select num).Count();

Console.WriteLine("No. of elements are " + count);

// Max no

var max = (from x in num

select x).Max();

// Max no

var avaerage = (from x in num

select x).Average();

}

}

using System;

using System.Collections.Generic;

using System.Linq;

class Program

{

static void Main()

{

List<int> list = new List<int>();

//list.Add(1);

//list.Add(20);

//list.Add(20);

// Collection Initializer

list = new List<int>()

{ 10 , 20 , 30 , 40 , 50, 19, 90};

var data = (from x in list

select x);

foreach(var x in data)

Console.WriteLine(x);

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

}

var num = 10;

// num = "aaa";

dynamic num1 = 10;

num1 = "aaa";

var sets the variable type at compile time

dynamic sets the variable type at run time