Dhruvil Dobariya

dhruvildobariya21@gmail.com

Abstract

[Draw your reader in with an engaging abstract. It is typically a short summary of the document.   
When you’re ready to add your content, just click here and start typing.]

C# Advanced

Documentation



Index

[1 Delegate 1](#_Toc133317928)

[1.1 Introduction 1](#_Toc133317929)

[1.2 Delegate: 1](#_Toc133317930)

[1.3 Multicast Delegate 2](#_Toc133317931)

[1.4 Return and Out parameter 3](#_Toc133317932)

[1.5 Async Delegate 5](#_Toc133317933)

[1.6 Function as a Argument 6](#_Toc133317934)

[1.7 Event 10](#_Toc133317935)

[2 Base Library 12](#_Toc133317936)

[2.1 Introduction 12](#_Toc133317937)

[2.2 Reflection 12](#_Toc133317938)

[2.3 Threading 13](#_Toc133317939)

[2.4 Net 15](#_Toc133317940)

[2.5 Web 18](#_Toc133317941)

[3 Lambda Expression 22](#_Toc133317942)

[3.1 Introduction 22](#_Toc133317943)

[4 Extension Method 24](#_Toc133317944)

[4.1 Introduction 24](#_Toc133317945)

[5 Linq 26](#_Toc133317946)

[5.1 Introduction 26](#_Toc133317947)

[5.2 Projection: 27](#_Toc133317948)

[5.3 Filtering 27](#_Toc133317949)

[5.4 Aggregate 27](#_Toc133317950)

[5.5 Sorting 28](#_Toc133317951)

[5.6 Quantifier 28](#_Toc133317952)

[5.7 Join 28](#_Toc133317953)

[5.8 Set 29](#_Toc133317954)

[5.9 Element 29](#_Toc133317955)

[5.10 Partition 30](#_Toc133317956)

[5.11 Concatenation 30](#_Toc133317957)

[5.12 Equality 30](#_Toc133317958)

[5.13 Conversation 30](#_Toc133317959)

[5.14 Generation 31](#_Toc133317960)

[6 ORM Tool 32](#_Toc133317961)

[6.1 Introduction 32](#_Toc133317962)

[6.2 ORM Lite 33](#_Toc133317963)

[7 Dynamic Type 36](#_Toc133317964)

[7.1 Introduction 36](#_Toc133317965)

[8 Background Worker 38](#_Toc133317966)

[8.1 Introduction 38](#_Toc133317967)

1. Delegate

# Introduction

* Delegate is a type which is used to represent reference of method.
* Namespace: **“System.Delegate”**.

# Delegate:

**Syntax:**

*delegate* <*return* type> <*delegate*-name> <parameter *list*>

* We can invoke delegate two way.

**Example:**

*namespace* DelegateLearn

{

*public* *class* Program

    {

*public* *delegate* *void* Calculation(*int* *a*, *int* *b*);

*public* *static* *void* Main(*string*[] *args*)

        {

*// Create Instance*

*// Method 1*

*//Calculation calculation = new Calculation(Addition);*

*// Method 2*

*Calculation* calculation = Addition;

*// Invoke Delegates*

*// Method 1*

*//calculation.Invoke(10, 20);*

*// Method 2*

            calculation(10, 20);

            calculation = Subtraction;

            calculation(10, 20);

            calculation = Multiplication;

            calculation(10, 20);

            calculation = Division;

            calculation(10, 20);

        }

*public* *static* *void* Addition(*int* *a*, *int* *b*)

        {

            Console.WriteLine(*a* + *b*);

        }

*public* *static* *void* Subtraction(*int* *a*, *int* *b*)

        {

            Console.WriteLine(*a* - *b*);

        }

    }

}

Output:

30

-10

200

0

# Multicast Delegate

* Multicast delegate allows us to invoke more then one method when we called one instance of delegate.
* We are used “+=” to append method and “-=” to remove method from delegate instance.

**Example:**

*namespace* DelegateLearn

{

*public* *class* MulticastDelegate

    {

*delegate* *void* Calculation(*int* *x*, *int* *y*);

*public* *static* *void* Main(*string*[] *args*)

        {

*// delegate contain more then one reference of method*

*Calculation* calculation = Addition;

            calculation += Subtraction;

            calculation += Multiplication;

            calculation += Division;

*// += Subscribe or append*

            calculation(20, 10);

*// remove subscription*

            calculation -= Subtraction;

            calculation(20, 10);

        }

*public* *static* *void* Addition(*int* *a*, *int* *b*)

        {

            Console.WriteLine(*a* + *b*);

        }

*public* *static* *void* Subtraction(*int* *a*, *int* *b*)

        {

            Console.WriteLine(*a* - *b*);

        }

*public* *static* *void* Multiplication(*int* *a*, *int* *b*)

        {

            Console.WriteLine(*a* \* *b*);

        }

*public* *static* *void* Division(*int* *a*, *int* *b*)

        {

            Console.WriteLine(*a* / *b*);

        }

    }

}

Output:

30

10

200

2

30

200

2

# Return and Out parameter

* In single cast delegate thing is easy to understand, because we are calling only one method at one instance of delegate.
* But in multicast we are calling more then one method at single instance of delegate.
* So how we decide which method’s return value of out param we get at the end.
* So the thing is we get only one return value or out param of one method which append last in instance of delegate.

**Example of Return:**

*namespace* DelegateLearn

{

*public* *class* MutlicastWithReturnType

    {

*public* *delegate* *int* Calculation(*int* *a*);

*public* *static* *void* Main(*string*[] *args*)

        {

*Calculation* calculation = AddFive;

            calculation += AddSeven;

*// It will return only last method's return value.*

            Console.WriteLine(calculation(3));

        }

*public* *static* *int* AddFive(*int* *a*)

        {

*return* *a* + 5;

        }

*public* *static* *int* AddSeven(*int* *a*)

        {

*return* *a* + 7;

        }

    }

}

Output:

10

**Example of Out Param:**

*namespace* DelegateLearn

{

*public* *class* MulticastWithOutputParameter

    {

*public* *delegate* *void* Calculation(*out* *int* *a*);

*public* *static* *void* Main(*string*[] *args*)

        {

*Calculation* calculation = GetTan;

            calculation += GetTwentyOne;

*// It will give only last method's out parameter value.*

*int* x;

            calculation(*out* x);

            Console.WriteLine(x);

        }

*public* *static* *void* GetTan(*out* *int* *a*)

        {

*a* = 10;

        }

*public* *static* *void* GetTwentyOne(*out* *int* *a*)

        {

*a* = 21;

        }

    }

}

Output:

21

# Async Delegate

* If any task take long time to execution then it’s hang the program until execution of that task complete.
* So using async delegate we can call delegate asynchronously.

**Example:**

*namespace* AsyncDelegate

{

*public* *class* Solution2

    {

*// using async await*

*delegate* *Task*<*int*> Calculation(*int* a, *int* b);

*public* *static* *void* Main(*string*[] args)

        {

            Console.WriteLine("Program start");

*Calculation* calculation = Sum;

            Console.WriteLine("Control going to the Sum method");

*Task*<*int*> result = calculation.Invoke(10, 20);

            Console.WriteLine("Control back to the Main method");

            Console.WriteLine(result.Result);

            Console.WriteLine("Program end");

        }

*public* *async* *static* *Task*<*int*> Sum(*int* a, *int* b)

        {

*return* *await* Task.Run(() =>

            {

                Console.WriteLine("Sum method running in background...");

                Thread.Sleep(10000);

*int* x = a + b;

                Console.WriteLine("Sum method running end");

*return* x;

            });

        }

    }

}

Output:

Program start

Control going to the Sum method

Control back to the Main method

Sum method running in background...

Sum method running end

30

Program end

# Function as a Argument

* We have many way to pass function as a argument in method.
  + Normal Delegate
  + Func Delegate
  + Action Delegate
  + Predicate Delegate

## Normal Delegate:

* Here we are just create delegate of method and pass delegate as argument.

**Example:**

*namespace* FunctionArgumentLearn

{

*public* *class* Delagate

    {

*public* *delegate* *int* Calculation(*int* *a*, *int* *b*);

*public* *static* *void* Main(*string*[] *args*)

        {

*Calculation* calculation = *new* *Calculation*(Sum);

            Console.WriteLine(AddNInCalculation(calculation, 10, 20, 40));

        }

*public* *static* *int* Sum(*int* *a*, *int* *b*)

        {

*return* *a* + *b*;

        }

*public* *static* *int* AddNInCalculation(*Calculation* *del*, *int* *n*, *int* *a*, *int* *b*)

        {

*return* *n* + *del*(*a*, *b*);

        }

    }

}

Output:

70

## Func Delegate:

* It is predefine delegate, which is use to pass method as a argument.

**Example:**

*namespace* FunctionArgumentLearn

{

*public* *class* Fun

    {

*// it is predefine delegate*

*public* *static* *void* Main(*string*[] *args*)

        {

            Console.WriteLine(AddNInCalculation(Sum, 10, 20, 40));

        }

*public* *static* *int* Sum(*int* *a*, *int* *b*)

        {

*return* *a* + *b*;

        }

*public* *static* *int* AddNInCalculation(*Func*<*int*, *int*, *int*> *function*, *int* *n*, *int* *a*, *int* *b*)

        {

*// Func<returntype, typeofarg1, typeofarg2, ...>*

*return* *n* + *function*(*a*, *b*);

        }

    }

}

Output:

70

## Action Delegate:

* It is also predefine delegate which is used to pass method as a argument.
* But it’s only used when our method return void which we pass as argument.

**Example:**

*namespace* FunctionArgumentLearn

{

*public* *class* Action

    {

*// it is predefine delegate*

*public* *static* *void* Main(*string*[] *args*)

        {

            WarpSum(Sum, 10, 20);

        }

*public* *static* *void* Sum(*int* *a*, *int* *b*)

        {

            Console.WriteLine(*a* + *b*);

        }

*public* *static* *void* WarpSum(*Action*<*int*, *int*> *function*, *int* *a*, *int* *b*)

        {

*// Action<typeofarg1, typeofarg2, ...>*

*// Action only used for these which don't return anything.*

*function*(*a*, *b*);

            Console.WriteLine("Function Run");

        }

    }

}

Output:

30

Function Run

## Predicate Delegate:

* It is also used for pass a method as a argument.
* But it also used when our argument method return boolean value.

**Example:**

*namespace* FunctionArgumentLearn

{

*public* *class* Predicate

    {

*// it is predefine delegate*

*// it only use to predict value n boolean*

*public* *static* *void* Main(*string*[] *args*)

        {

            Console.WriteLine($"Is odd: {Check(21, IsOdd)}");

            Console.WriteLine($"Is even: {Check(21, IsEven)}");

        }

*public* *static* *bool* Check(*int* *a*, *Predicate*<*int*> *predicatemethod*)

        {

*return* *predicatemethod*(*a*);

        }

*public* *static* *bool* IsEven(*int* *a*)

        {

*if* (*a* % 2 == 0)

            {

*return* *true*;

            }

*return* *false*;

        }

*public* *static* *bool* IsOdd(*int* *a*)

        {

*if* (*a* % 2 == 0)

            {

*return* *false*;

            }

*return* *true*;

        }

    }

}

Output:

Is odd: True

Is even: False

# Event

* In multicast delegate we can append, remove or redefine instance of delegate.
* But the when we append or remove any method in instance of delegate using “+=” or “-=” respectively, it may chances to write “=” by mistake.
* If we write “=” then all append method remove, which we append previously, and we can’t get our desirable output and it’s generate bug.
* So solve this problem events come in picture.
* Event is a encapsulated version of delegate.
* It provide publish subscriber mode.
* In event we can only use “+=” for subscribe event and “-=” for unsubscribe event, we can’t redefine event using “=” like multicast delegate.

**Example:**

*namespace* EventLearn

{

*public* *class* Solution

    {

*// event should solve this problem.*

*// it encapsulate delegate and it only use publish and subscribe.*

*// We can only use += or -= not only =.*

*// Event Handlers can't return a value. They are always void.*

*public* *delegate* *void* Calculation(*int* *x*, *int* *y*);

*public* *event* *Calculation* OnCalculation = *null*;

*public* *static* *void* Main(*string*[] *args*)

        {

*// Event only use subscribe unsubscribe method.*

*// We cant use = operator for new instance.*

*Solution* solution = *new* *Solution*();

            solution.OnCalculation += Addition;

            solution.OnCalculation += Subtraction;

*//solution.OnCalculation = null; // throw an error.*

            solution.OnCalculation -= Subtraction;

            solution.OnCalculation += Division;

            solution.OnCalculation(10, 20);

        }

*public* *static* *void* Addition(*int* *a*, *int* *b*)

        {

            Console.WriteLine($"Addition of {*a*} and {*b*} is : {*a* + *b*}");

        }

*public* *static* *void* Subtraction(*int* *a*, *int* *b*)

        {

            Console.WriteLine($"Subtraction of {*a*} and {*b*} is : {*a* - *b*}");

        }

*public* *static* *void* Multiplication(*int* *a*, *int* *b*)

        {

            Console.WriteLine($"Multiplication of {*a*} and {*b*} is : {*a* \* *b*}");

        }

*public* *static* *void* Division(*int* *a*, *int* *b*)

        {

            Console.WriteLine($"Division of {*a*} and {*b*} is : {*a* / *b*}");

        }

    }

}

Output:

Addition of 10 and 20 is : 30 Division of 10 and 20 is : 0

1. Base Library

# Introduction

* We have many base libraries in C#.
  + System.Reflection
  + System.Threading
  + System.Net
  + System.Web

# Reflection

* Reflection is used to get type information and metadata about any object at runtime.
* It’s namespace is System.Reflection
* It allows view attribute information at runtime.
* It allows examining various types in an assembly and instantiate these types.
* It allows late binding to methods and properties.
* It allows creating new types at runtime and then performs some tasks using those types.

**Properties:**

|  |  |
| --- | --- |
| Property | Description |
| Assembly | Gets the Assembly for this type. |
| AssemblyQualifiedName | Gets the Assembly qualified name for this type. |
| Attributes | Gets the Attributes associated with the type. |
| BaseType | Gets the base or parent type. |
| FullName | Gets the fully qualified name of the type. |
| IsAbstract | is used to check if the type is Abstract. |
| IsArray | is used to check if the type is Array. |
| IsClass | is used to check if the type is Class. |
| IsEnum | is used to check if the type is Enum. |
| IsInterface | is used to check if the type is Interface. |
| IsNested | is used to check if the type is Nested. |
| IsPrimitive | is used to check if the type is Primitive. |
| IsPointer | is used to check if the type is Pointer. |
| IsNotPublic | is used to check if the type is not Public. |
| IsPublic | is used to check if the type is Public. |
| IsSealed | is used to check if the type is Sealed. |
| IsSerializable | is used to check if the type is Serializable. |
| MemberType | is used to check if the type is Member type of Nested type. |
| Module | Gets the module of the type. |
| Name | Gets the name of the type. |
| Namespace | Gets the namespace of the type. |

**Methods:**

|  |  |
| --- | --- |
| Method | Description |
| GetConstructors() | Returns all the public constructors for the Type. |
| GetConstructors(BindingFlags) | Returns all the constructors for the Type with specified BindingFlags. |
| GetFields() | Returns all the public fields for the Type. |
| GetFields(BindingFlags) | Returns all the public constructors for the Type with specified BindingFlags. |
| GetMembers() | Returns all the public members for the Type. |
| GetMembers(BindingFlags) | Returns all the members for the Type with specified BindingFlags. |
| GetMethods() | Returns all the public methods for the Type. |
| GetMethods(BindingFlags) | Returns all the methods for the Type with specified BindingFlags. |
| GetProperties() | Returns all the public properties for the Type. |
| GetProperties(BindingFlags) | Returns all the properties for the Type with specified BindingFlags. |
| GetType() | Gets the current Type. |
| GetType(String) | Gets the Type for the given name. |

# Threading

* It help use to create thread in and control on current thread.
* It’s namespace is System.Threading

**Properties:**

|  |  |
| --- | --- |
| 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. |

**Methods:**

|  |  |
| --- | --- |
| 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. |

## Life cycle of Thread:

* Thread has five state in life cycle,
  + Unstarted
  + Runnable (Ready to run)
  + Running
  + Not Runnable
  + Dead (Terminate)

**Unstarted State:**

* When the instance of Thread class is created, it is in unstarted state by default.

**Runnable State:**

* When start() method on the thread is called, it is in runnable or ready to run state.

**Running State**

* Only one thread within a process can be executed at a time.
* At the time of execution, thread is in running state.

**Not Runnable State:**

* The thread is in not runnable state, if sleep() or wait() method is called on the thread, or input/output operation is blocked.

**Dead State:**

* After completing the task, thread enters into dead or terminated state.

# Net

* It helps us to get information about request and response of network protocols.
* It’s namespace is System.Net
* We have many class in System.Net namespace,
  + HttpClient
  + HttpResponceMessage
  + SmtpClient

## HttpClient:

* It is used to request using http protocol.

**Properties:**

|  |  |
| --- | --- |
| Properties | Methods |
| [BaseAddress](https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpclient.baseaddress?view=net-8.0#system-net-http-httpclient-baseaddress) | Gets or sets the base address of Uniform Resource Identifier (URI) of the Internet resource used when sending requests. |
| [DefaultProxy](https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpclient.defaultproxy?view=net-8.0#system-net-http-httpclient-defaultproxy) | Gets or sets the global Http proxy. |
| [DefaultRequestHeaders](https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpclient.defaultrequestheaders?view=net-8.0#system-net-http-httpclient-defaultrequestheaders) | Gets the headers which should be sent with each request. |
| [DefaultRequestVersion](https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpclient.defaultrequestversion?view=net-8.0#system-net-http-httpclient-defaultrequestversion) | Gets or sets the default HTTP version used on subsequent requests made by this [HttpClient](https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpclient?view=net-8.0) instance. |
| [DefaultVersionPolicy](https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpclient.defaultversionpolicy?view=net-8.0#system-net-http-httpclient-defaultversionpolicy) | Gets or sets the default version policy for implicitly created requests in convenience methods, for example, [GetAsync(String)](https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpclient.getasync?view=net-8.0" \l "system-net-http-httpclient-getasync(system-string)) and [PostAsync(String, HttpContent)](https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpclient.postasync?view=net-8.0" \l "system-net-http-httpclient-postasync(system-string-system-net-http-httpcontent)). |
| [MaxResponseContentBufferSize](https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpclient.maxresponsecontentbuffersize?view=net-8.0#system-net-http-httpclient-maxresponsecontentbuffersize) | Gets or sets the maximum number of bytes to buffer when reading the response content. |
| [Timeout](https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpclient.timeout?view=net-8.0#system-net-http-httpclient-timeout) | Gets or sets the timespan to wait before the request times out. |

**Methods:**

|  |  |
| --- | --- |
| Properties | Methods |
| CancelPendingRequests | Cancel all pending requests on this instance |
| DeleteAsync | Send a DELETE request to the specified Uri as an asynchronous operation |
| Dispose(Boolean) | Releases the unmanaged resources used by the HttpClient and optionally disposes of the managed resources. |
| GetAsync | Send a GET request to the specified Uri as an asynchronous operation |
| GetByteArrayAsync | Send a GET request to the specified Uri and return the response body as a byte array in an asynchronous operation |
| GetStreamAsync | Send a GET request to the specified Uri and return the response body as a stream in an asynchronous operation |
| GetStringAsync | Send a GET request to the specified Uri and return the response body as a string in an asynchronous operation |
| PatchAsync | Sends a PATCH request to a Uri designated as a string as an asynchronous operation |
| PostAsync | Send a POST request to the specified Uri as an asynchronous operation. |
| PutAsync | Send a PUT request to the specified Uri as an asynchronous operation |
| Send | Sends an HTTP request with the specified request |
| SendAsync | Send an HTTP request as an asynchronous operation |

## HttpResponceMessage:

* It is used to handle response message of http request.

**Properties:**

|  |  |
| --- | --- |
| Properties | Description |
| [Content](https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpresponsemessage.content?view=net-8.0#system-net-http-httpresponsemessage-content) | Gets or sets the content of a HTTP response message. |
| [Headers](https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpresponsemessage.headers?view=net-8.0#system-net-http-httpresponsemessage-headers) | Gets the collection of HTTP response headers. |
| [IsSuccessStatusCode](https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpresponsemessage.issuccessstatuscode?view=net-8.0#system-net-http-httpresponsemessage-issuccessstatuscode) | Gets a value that indicates if the HTTP response was successful. |
| [ReasonPhrase](https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpresponsemessage.reasonphrase?view=net-8.0#system-net-http-httpresponsemessage-reasonphrase) | Gets or sets the reason phrase which typically is sent by servers together with the status code. |
| [RequestMessage](https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpresponsemessage.requestmessage?view=net-8.0#system-net-http-httpresponsemessage-requestmessage) | Gets or sets the request message which led to this response message. |
| [StatusCode](https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpresponsemessage.statuscode?view=net-8.0#system-net-http-httpresponsemessage-statuscode) | Gets or sets the status code of the HTTP response. |
| [TrailingHeaders](https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpresponsemessage.trailingheaders?view=net-8.0#system-net-http-httpresponsemessage-trailingheaders) | Gets the collection of trailing headers included in an HTTP response. |
| [Version](https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpresponsemessage.version?view=net-8.0#system-net-http-httpresponsemessage-version) | Gets or sets the HTTP message version. |

**Methods:**

|  |  |
| --- | --- |
| Method | Description |
| [Dispose()](https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpresponsemessage.dispose?view=net-8.0#system-net-http-httpresponsemessage-dispose) | Releases the unmanaged resources and disposes of unmanaged resources used by the [HttpResponseMessage](https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpresponsemessage?view=net-8.0). |
| [Dispose(Boolean)](https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpresponsemessage.dispose?view=net-8.0#system-net-http-httpresponsemessage-dispose(system-boolean)) | Releases the unmanaged resources used by the [HttpResponseMessage](https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpresponsemessage?view=net-8.0) and optionally disposes of the managed resources. |
| [EnsureSuccessStatusCode()](https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpresponsemessage.ensuresuccessstatuscode?view=net-8.0#system-net-http-httpresponsemessage-ensuresuccessstatuscode) | Throws an exception if the [IsSuccessStatusCode](https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpresponsemessage.issuccessstatuscode?view=net-8.0" \l "system-net-http-httpresponsemessage-issuccessstatuscode) property for the HTTP response is false. |
| [Equals(Object)](https://learn.microsoft.com/en-us/dotnet/api/system.object.equals?view=net-8.0#system-object-equals(system-object)) | Determines whether the specified object is equal to the current object.  (Inherited from [Object](https://learn.microsoft.com/en-us/dotnet/api/system.object?view=net-8.0)) |
| [GetHashCode()](https://learn.microsoft.com/en-us/dotnet/api/system.object.gethashcode?view=net-8.0#system-object-gethashcode) | Serves as the default hash function.  (Inherited from [Object](https://learn.microsoft.com/en-us/dotnet/api/system.object?view=net-8.0)) |
| [GetType()](https://learn.microsoft.com/en-us/dotnet/api/system.object.gettype?view=net-8.0#system-object-gettype) | Gets the [Type](https://learn.microsoft.com/en-us/dotnet/api/system.type?view=net-8.0) of the current instance.  (Inherited from [Object](https://learn.microsoft.com/en-us/dotnet/api/system.object?view=net-8.0)) |
| [MemberwiseClone()](https://learn.microsoft.com/en-us/dotnet/api/system.object.memberwiseclone?view=net-8.0#system-object-memberwiseclone) | Creates a shallow copy of the current [Object](https://learn.microsoft.com/en-us/dotnet/api/system.object?view=net-8.0).  (Inherited from [Object](https://learn.microsoft.com/en-us/dotnet/api/system.object?view=net-8.0)) |
| [ToString()](https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpresponsemessage.tostring?view=net-8.0#system-net-http-httpresponsemessage-tostring) | Returns a string that represents the current object. |

## SmtpClient:

* It is used to request using smtp protocol.

**Properties:**

|  |  |
| --- | --- |
| Properties | Description |
| [ClientCertificates](https://learn.microsoft.com/en-us/dotnet/api/system.net.mail.smtpclient.clientcertificates?view=net-8.0#system-net-mail-smtpclient-clientcertificates) | Specify which certificates should be used to establish the Secure Sockets Layer (SSL) connection. |
| [Credentials](https://learn.microsoft.com/en-us/dotnet/api/system.net.mail.smtpclient.credentials?view=net-8.0#system-net-mail-smtpclient-credentials) | Gets or sets the credentials used to authenticate the sender. |
| [DeliveryFormat](https://learn.microsoft.com/en-us/dotnet/api/system.net.mail.smtpclient.deliveryformat?view=net-8.0#system-net-mail-smtpclient-deliveryformat) | Gets or sets the delivery format used by [SmtpClient](https://learn.microsoft.com/en-us/dotnet/api/system.net.mail.smtpclient?view=net-8.0) to send email. |
| [DeliveryMethod](https://learn.microsoft.com/en-us/dotnet/api/system.net.mail.smtpclient.deliverymethod?view=net-8.0#system-net-mail-smtpclient-deliverymethod) | Specifies how outgoing email messages will be handled. |
| [EnableSsl](https://learn.microsoft.com/en-us/dotnet/api/system.net.mail.smtpclient.enablessl?view=net-8.0#system-net-mail-smtpclient-enablessl) | Specify whether the [SmtpClient](https://learn.microsoft.com/en-us/dotnet/api/system.net.mail.smtpclient?view=net-8.0) uses Secure Sockets Layer (SSL) to encrypt the connection. |
| [Host](https://learn.microsoft.com/en-us/dotnet/api/system.net.mail.smtpclient.host?view=net-8.0#system-net-mail-smtpclient-host) | Gets or sets the name or IP address of the host used for SMTP transactions. |
| [PickupDirectoryLocation](https://learn.microsoft.com/en-us/dotnet/api/system.net.mail.smtpclient.pickupdirectorylocation?view=net-8.0#system-net-mail-smtpclient-pickupdirectorylocation) | Gets or sets the folder where applications save mail messages to be processed by the local SMTP server. |
| [Port](https://learn.microsoft.com/en-us/dotnet/api/system.net.mail.smtpclient.port?view=net-8.0#system-net-mail-smtpclient-port) | Gets or sets the port used for SMTP transactions. |
| [ServicePoint](https://learn.microsoft.com/en-us/dotnet/api/system.net.mail.smtpclient.servicepoint?view=net-8.0#system-net-mail-smtpclient-servicepoint) | Gets the network connection used to transmit the email message. |
| [TargetName](https://learn.microsoft.com/en-us/dotnet/api/system.net.mail.smtpclient.targetname?view=net-8.0#system-net-mail-smtpclient-targetname) | Gets or sets the Service Provider Name (SPN) to use for authentication when using extended protection. |
| [Timeout](https://learn.microsoft.com/en-us/dotnet/api/system.net.mail.smtpclient.timeout?view=net-8.0#system-net-mail-smtpclient-timeout) | Gets or sets a value that specifies the amount of time after which a synchronous [Send](https://learn.microsoft.com/en-us/dotnet/api/system.net.mail.smtpclient.send?view=net-8.0) call times out. |
| [UseDefaultCredentials](https://learn.microsoft.com/en-us/dotnet/api/system.net.mail.smtpclient.usedefaultcredentials?view=net-8.0#system-net-mail-smtpclient-usedefaultcredentials) | Gets or sets a [Boolean](https://learn.microsoft.com/en-us/dotnet/api/system.boolean?view=net-8.0) value that controls whether the [DefaultCredentials](https://learn.microsoft.com/en-us/dotnet/api/system.net.credentialcache.defaultcredentials?view=net-8.0" \l "system-net-credentialcache-defaultcredentials) are sent with requests. |

**Methods:**

|  |  |
| --- | --- |
| Properties | Description |
| Dispose | Sends a QUIT message to the SMTP server, gracefully ends the TCP connection, and releases all resources used by the current instance of the SmtpClient class. |
| OnSendCompleted | Raises the SendCompleted event |
| Send | Sends an email message to an SMTP server for delivery. These methods block while the message is being transmitted |
| SendAsync | Sends an email message. These methods do not block the calling thread |
| SendAsyncCancel | Cancels an asynchronous operation to send an email message |
| SendMailAsync | Sends the specified message to an SMTP server for delivery as an asynchronous operation. |

# Web

* It help us to do web related task.
* It’s namespace is System.Web
* We have many class in System.Web namespace,
  + HttpContext
  + HttpRequest
  + HttpResponce
  + User
  + Session
  + WebSokets

## HttpContext:

**Properties:**

|  |  |
| --- | --- |
| Properties | Description |
| Request | Gets or sets the HttpRequest object for the current request |
| Request | Gets or sets the HttpResponse object for the current response |
| User | Gets or sets the ClaimsPrincipal object representing the current user |
| Items | Gets a key-value dictionary that can be used to store and share data during the lifetime of the current request |
| Session | Gets the ISession object for the current session. |
| Connection | Gets the ConnectionInfo object for the current connection |
| WebSockets | Gets the WebSocketManager object for the current connection if it is a WebSocket connection |

**Methods:**

|  |  |
| --- | --- |
| Method | Description |
| Abort() | Aborts the current connection. |
| AuthenticateAsync() | Authenticates the current request |
| ChallengeAsync() | Challenges the current request with a specific authentication scheme |
| ForbidAsync() | Forbids the current request with a specific authentication scheme |
| GetEndpoint() | Gets the Endpoint object for the current request |
| SignOutAsync() | Signs out the current user |
| TryGetFeature() | Tries to get the specified feature from the HttpContext.Features collection |

## HttpRequest:

**Properties:**

|  |  |
| --- | --- |
| Properties | Description |
| ContentType | Gets the content type of the request body, if any |
| Headers | Gets the collection of headers in the request |
| Host | Gets the host and port information from the request header |
| IsHttps | Gets a value that indicates whether the request is an HTTPS request |
| Method | Gets or sets the HTTP method used for the request |
| Query | Gets the query string values as a collection of key-value pairs |
| QueryString | Gets the query string values as a string |
| Path | Gets the path of the request |
| PathBase | Gets the base path of the request |
| Protocol | Gets or sets the protocol used for the request |
| Scheme | Gets or sets the URI scheme used for the request |

**Methods:**

|  |  |
| --- | --- |
| Method | Description |
| Body() | Gets the request body as a stream |
| GetDisplayUrl() | Gets the complete request URL, including the query string and host |
| GetTypedHeaders() | Gets the strongly typed HTTP request headers |
| ReadFormAsync() | Reads the form values from the request body |

## HttpResponce:

**Properties:**

|  |  |
| --- | --- |
| Properties | Description |
| ContentType | Gets or sets the content type of the response |
| Headers | Gets the collection of headers in the response |
| StatusCode | Gets or sets the HTTP status code of the response |
| Body | Gets or sets the response body as a stream |

**Methods:**

|  |  |
| --- | --- |
| Method | Description |
| WriteAsync() | Writes a string to the response body |
| WriteAsJsonAsync() | Serializes an object to JSON and writes it to the response body |
| Redirect() | Redirects the client to a new URL |
| OnStarting() | Registers an action to be executed just before the response starts |
| OnCompleted() | Registers an action to be executed after the response has completed |

## User:

**Properties:**

|  |  |
| --- | --- |
| Properties | Description |
| Identity | Gets the ClaimsIdentity object representing the user's identity |
| Claims | Gets the collection of claims associated with the user |
| Identity.IsAuthenticated | Gets a value indicating whether the user's identity has been authenticated |
| Identity.Name | Gets the name of the user |

**Methods:**

|  |  |
| --- | --- |
| Method | Description |
| FindFirst() | Finds the first claim of the specified type |
| FindAll() | Finds all claims of the specified type |
| HasClaim() | Determines whether the user has a claim of the specified type and value |
| IsInRole() | Determines whether the user belongs to the specified role |

## Session:

**Properties:**

|  |  |
| --- | --- |
| Properties | Description |
| Id | Gets the unique identifier for the current session |
| IsAvailable | Gets a value indicating whether the session is available |
| Keys | Gets a collection of keys for all session items |
| Timeout | Gets or sets the timeout period for the session |

**Methods:**

|  |  |
| --- | --- |
| Method | Description |
| Set() | Sets the value of a session item |
| Get() | Gets the value of a session item |
| Remove() | Removes a session item |
| Clear() | Clears all session items |
| LoadAsync() | Loads the session data from the session store |

## WebSocket:

**Properties:**

|  |  |
| --- | --- |
| Properties | Description |
| ConnectionId | Gets the unique identifier for the connection |
| User | Gets or sets the user associated with the connection |
| Transport | Gets the transport used by the connection |
| Features | Gets a collection of features associated with the connection |

**Methods:**

|  |  |
| --- | --- |
| Method | Description |
| StartAsync() | Starts the connection |
| StopAsync() | Stops the connection |
| SendAsync() | Sends data to the client over the connection |
| DisposeAsync() | Disposes the connection asynchronously |

1. Lambda Expression

# Introduction

* Lambda expressions in C# are a shorthand syntax for creating anonymous functions.
* These functions can be assigned to a delegate, used as a parameter for a method, or returned as a result of a method.

**Syntax:**

*// lambda expression*

(parameter\_list) => expression

*// Or*

(parameter\_list) =>

{

*// code...*

}

**Example:**

*namespace* LamdaExpressionLearn

{

*public* *class* Program1

    {

*public* *static* *void* Main(*string*[] args)

        {

*List*<*int*> list = *new* *List*<*int*>() { 1, 2, 3, 4, 5 };

*int* sumOfSqrt = list.Select(element => element \* element).Sum();

            Console.WriteLine(sumOfSqrt);

*// it's like:*

*// int sumOfSqrt = 0;*

*// foreach (int element in list)*

*// {*

*//    sumOfSqrt += element \* element;*

*// }*

*// Single line*

*Func*<*string*, *int*, *bool*> isLengthGreaterThenN = (x, y) => x.Length > y;

            Console.WriteLine(isLengthGreaterThenN("Dhruvil Dobariya", 10));

*// Multi line*

*Action*<*int*, *int*> sum = (x, y) =>

            {

*int* ans = x + y;

                Console.WriteLine($"Ans: {ans}");

            };

            sum(10, 20);

        }

    }

}

1. Extension Method

# Introduction

* Extension methods in C# allow you to add new methods to existing classes or interfaces without modifying their source code.
* Extension methods are defined as static methods in a static class.

**Syntax:**

*public* *static* <return\_type> <method\_name>(*this* <extended\_type>

<parameter\_name>, <additional\_parameters>)

{

*// Method implementation*

}

**Example:**

*namespace* ExtrensionMethodLearn

{

*public* *class* Program

    {

*public* *static* *void* Main(*string*[] args)

        {

*Calculation* calculation = *new* *Calculation*();

            Console.WriteLine(calculation.Addition(10, 20));

            Console.WriteLine(calculation.Sqrt(100));

        }

    }

*public* *class* Calculation

    {

*public* *int* Addition(*int* x, *int* y)

        {

*return* x + y;

        }

*public* *int* Subtraction(*int* x, *int* y)

        {

*return* x - y;

        }

*public* *int* Multiplication(*int* x, *int* y)

        {

*return* x \* y;

        }

*public* *int* Division(*int* x, *int* y)

        {

*return* x / y;

        }

    }

*public* *static* *class* CalculationExtention

    {

*public* *static* *double* Sqrt(*this* *Calculation* calculation, *int* x)

        {

*return* Math.Sqrt(x);

        }

*public* *static* *double* Percentage(*this* *Calculation* calculation, *int* x, *int* total)

        {

*return* (x \* 100) / total;

        }

    }

}

1. Linq

# Introduction

* LINQ (Language Integrated Query) is a feature in C# that allows you to write queries against collections of objects, databases, and other data sources.
* With LINQ, you can write declarative queries using a set of query operators that are integrated into the C# language syntax.
* LINQ provides a uniform way of querying different types of data sources, including arrays, lists, XML documents, SQL databases, and more.
* The LINQ query syntax resembles SQL, making it easy to learn for developers who are familiar with SQL.
* LINQ has two main syntaxes,
  + Query syntax
  + Method syntax

*namespace* LinqLearn

{

*public* *class* Program

    {

*public* *static* *void* Main(*string*[] args)

        {

*// Query Syntax*

*int*[] numbers = { 1, 2, 3, 4, 5 };

*var* evenNumbers = *from* n *in* numbers

*where* n % 2 == 0

*select* n;

*// Method Syntax*

*int*[] numbers = { 1, 2, 3, 4, 5 };

*var* evenNumbers = numbers.Where(n => n % 2 == 0);

        }

    }

}

* Linq contains many types of operations like,
  + Projection
  + Filtering
  + Aggregate
  + Sorting
  + Quantifier
  + Join
  + Set
  + Element
  + Partition
  + Concatenation
  + Equality
  + Conversation
  + Generation

# Projection:

* Projection operations in LINQ are used to transform the data returned by a query into a different format.

|  |  |
| --- | --- |
| Operator | Description |
| Select | Projects each element of a sequence into a new form by applying a projection function. The result is a new sequence of the projected elements. |
| SelectMany | Projects each element of a sequence to a new sequence, and then flattens the resulting sequences into a single sequence. |

# Filtering

* Filtering in LINQ is the process of selecting elements from a sequence that satisfy a given condition.

|  |  |
| --- | --- |
| Operator | Description |
| Where | Filters a sequence based on a specified condition. The operator returns a new sequence that contains only the elements of the original sequence that satisfy the condition. |
| OfType | Filters a sequence to include only elements of a specified type. The operator returns a new sequence that contains only the elements of the original sequence that are of the specified type. |

# Aggregate

|  |  |
| --- | --- |
| Operator | Description |
| GroupBy | Groups the elements of a sequence based on a specified key and returns a sequence of groupings. The groupings contain a key and a sequence of elements that share that key. |
| Having | Filters the groups produced by a GroupBy operation based on a specified condition. The Having operator is often used in conjunction with GroupBy to filter the groups based on aggregate values. |
| ToLookup | Groups the elements of a sequence based on a specified key and returns a lookup table. The lookup table contains a key and a sequence of elements that share that key. |
| Count | Returns the number of elements in a sequence. |
| Sum | Computes the sum of the numeric values in a sequence. |
| Min | Returns the minimum value in a sequence. |
| Max | Returns the maximum value in a sequence. |
| Average | Computes the average of the numeric values in a sequence. |

# Sorting

|  |  |
| --- | --- |
| Operator | Description |
| OrderBy | Sorts the elements of a sequence in ascending order based on a specified key. |
| OrderByDescending | Sorts the elements of a sequence in descending order based on a specified key. |
| ThenBy | Performs a secondary sort on the elements of a sequence in ascending order based on a specified key. This operator is typically used in conjunction with OrderBy. |
| ThenByDescending | Performs a secondary sort on the elements of a sequence in descending order based on a specified key. This operator is typically used in conjunction with OrderByDescending. |
| Reverse | Reverses the order of the elements in a sequence. |

# Quantifier

|  |  |
| --- | --- |
| Operator | Description |
| All | Determines whether all elements of a sequence satisfy a specified condition. Returns true if every element satisfies the condition; otherwise, false. |
| Any | Determines whether any element of a sequence satisfies a specified condition. Returns true if at least one element satisfies the condition; otherwise, false. |
| Contains | Determines whether a sequence contains a specified element. Returns true if the element is found; otherwise, false. |

# Join

|  |  |
| --- | --- |
| Operator | Description |
| Join | Returns a sequence of elements that have matching keys in two sequences, based on a specified key selector function and an optional element selector function. The result sequence contains elements of both sequences that have matching keys. |
| GroupJoin | Groups the elements of one sequence and the matching elements of another sequence into inner groups, based on a specified key selector function. The result sequence contains elements of the first sequence and their matching inner groups. |

# Set

|  |  |
| --- | --- |
| Operator | Description |
| Distinct | Returns a new sequence that contains unique elements from the original sequence, based on a specified equality comparer. |
| Except | Returns the set difference of two sequences, i.e., the elements that are in the first sequence but not in the second sequence, based on a specified equality comparer. |
| Intersect | Returns the set intersection of two sequences, i.e., the elements that are in both sequences, based on a specified equality comparer. |
| Union | Returns the set union of two sequences, i.e., the unique elements that are in either the first or the second sequence, based on a specified equality comparer. |

# Element

|  |  |
| --- | --- |
| Method | Description |
| ElementAt | Returns the element at a specified index in a sequence. Throws an exception if the index is out of range. |
| ElementAtOrDefault | Returns the element at a specified index in a sequence, or a default value if the index is out of range. |
| Find | Searches for an element that matches a specified condition in a sequence, and returns the first matching element, or null if no element is found. |
| FindLast | Searches for an element that matches a specified condition in a sequence, and returns the last matching element, or null if no element is found. |
| FindAll | Returns a new sequence that contains all elements from the original sequence that match a specified condition. |
| FindIndex | Returns the index of the first element that matches a specified condition in a sequence, or -1 if no element is found. |
| FindLastIndex | Returns the index of the last element that matches a specified condition in a sequence, or -1 if no element is found. |
| First | Returns the first element in a sequence that matches a specified condition. Throws an exception if no element is found. |
| FirstOrDefault | Returns the first element in a sequence that matches a specified condition, or a default value if no element is found. |
| Last | Returns the last element in a sequence that matches a specified condition. Throws an exception if no element is found. |
| LastOrDefault | Returns the last element in a sequence that matches a specified condition, or a default value if no element is found. |
| IndexOf | Returns the index of the first occurrence of a specified element in a sequence, or -1 if the element is not found. |
| LastIndexOf | Returns the index of the last occurrence of a specified element in a sequence, or -1 if the element is not found. |
| Single | Returns the only element in a sequence that matches a specified condition. Throws an exception if there is not exactly one element that matches the condition. |
| SingleOrDefault | Returns the only element in a sequence that matches a specified condition, or a default value if no element is found. Throws an exception if there is more than one element that matches the condition. |

# Partition

|  |  |
| --- | --- |
| Method | Description |
| Take | Returns a specified number of contiguous elements from the start of a sequence. |
| TakeLast | Returns a specified number of contiguous elements from the end of a sequence. |
| TakeWhile | Returns elements from a sequence as long as a specified condition is true, starting from the beginning of the sequence. |
| Skip | Skips a specified number of contiguous elements from the start of a sequence, and returns the remaining elements. |
| SkipLast | Skips a specified number of contiguous elements from the end of a sequence, and returns the remaining elements. |
| SkipWhile | Skips elements from a sequence as long as a specified condition is true, starting from the beginning of the sequence, and returns the remaining elements. |

# Concatenation

|  |  |
| --- | --- |
| Method | Description |
| Concat | Concatenates two sequences. Returns a new sequence that contains all the elements from the first sequence, followed by all the elements from the second sequence. |

# Equality

|  |  |
| --- | --- |
| Method | Description |
| SequenceEqual | Determines whether two sequences are equal. Returns true if the two sequences have the same number of elements and the corresponding elements in the two sequences are equal, as determined by the default equality comparer for their type. Otherwise, returns false. |

# Conversation

|  |  |
| --- | --- |
| Method | Description |
| AsEnumerable | Returns the input sequence as an IEnumerable<T>. This method has no effect on sequences that are already IEnumerable<T>. |
| AsQueryable | Converts the input sequence to an IQueryable<T>. This allows for the use of deferred execution and enables the use of query providers such as LINQ to SQL or Entity Framework. |
| Cast | Casts the elements of a sequence to a specified type. |
| ToArray | Converts the elements of a sequence to an array. |
| ToDictionary | Creates a Dictionary<TKey, TValue> from a sequence of elements according to a specified key selector function. |
| ToList | Creates a List<T> from a sequence of elements. |

# Generation

|  |  |
| --- | --- |
| Method | Description |
| DefaultIfEmpty | Returns the input sequence if it is not empty. Otherwise, returns a sequence containing a single default value of the same type as the elements in the input sequence. |
| Empty | Returns an empty sequence of a specified type. |
| Range | Generates a sequence of integers within a specified range. |
| Repeat | Generates a sequence that contains one repeated value. |

1. ORM Tool

# Introduction

* ORM stands for Object-Relational Mapping.
* It is a programming technique that enables developers to map objects in object-oriented programming languages, such as C# or Java, to relational database tables.
* ORM frameworks provide a set of tools for developers to work with databases using objects and abstract away many of the low-level details of interacting with a database, such as constructing and executing SQL queries.
* Some popular ORM frameworks for C# include Entity Framework, NHibernate, and Dapper.
* These frameworks provide a way for developers to interact with databases using object-oriented principles, which can make it easier to write and maintain code.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Feature | Entity Framework Core | NHibernate | Dapper | ORM Lite |
| Mapping | Code First, Database First, and Model First | XML, fluent API, and attribute-based | None | Code First and Database First |
| Performance | Slower than micro-ORMs like Dapper, but faster than full-featured ORMs | Slower than micro-ORMs like Dapper | Faster than full-featured ORMs like EF | Faster than full-featured ORMs like EF |
| Database Support | SQL Server, PostgreSQL, MySQL, SQLite, Oracle, and more | SQL Server, PostgreSQL, MySQL, SQLite, and more | Any database with a .NET data provider | SQL Server, Oracle, MySQL, and SQLite |
| Learning Curve | Moderate | Steep | Minimal | Minimal |
| Community Support | Large and active community | Large and active community | Active community | Small but growing community |
| Features | LINQ to Entities, lazy loading, change tracking, and more | LINQ to NHibernate, lazy loading, and more | None | Code First and Database First support |
| License | Open source (MIT License) | Open source (LGPL or Apache License) | Open source (MIT License) | Open source (MIT License) |
| Querying Capabilities | Supports complex queries through LINQ and SQL | Supports complex queries through LINQ and HQL | Basic query capabilities | Supports simple to moderately complex queries through LINQ and SQL |

# ORM Lite

* ORM Lite is a lightweight, open-source object-relational mapping (ORM) framework for .NET and Xamarin platforms.
* It provides a simple and intuitive API for mapping C# objects to database tables, and supports various database providers such as SQLite, MySQL, SQL Server, Oracle, PostgreSQL, and more.

**Features:**

* Simple and lightweight API
* Flexible mapping options
* Cross-platform support
* Performance oriented

**Different types of APIs:**

* ORM contains many types APIs like,
  + Select
  + Insert
  + Update
  + Delete
  + SQL

## Select API:

|  |  |
| --- | --- |
| Method | Description |
| Select | Projects each element of a sequence into a new form. |
| Single | Returns the only element of a sequence that satisfies a specified condition, and throws an exception if none or more than one such element exists. |
| SingleById | Returns the single element of a table that matches the specified ID. |
| SingleWhere | Returns the only element of a sequence that satisfies a specified condition, and throws an exception if none or more than one such element exists. |
| Where | Filters a sequence of values based on a predicate. |
| From | Specifies the data source for a LINQ query. |
| Lookup | Groups the elements of a sequence according to a specified key selector function and returns a sequence of key-grouping pairs. |
| Dictionary | Creates a dictionary from a sequence according to a specified key selector function. |
| OrderBy | Sorts the elements of a sequence in ascending order according to a key. |
| OrderByDescending | Sorts the elements of a sequence in descending order according to a key. |
| Join | Correlates the elements of two sequences based on matching keys. |
| GroupBy | Groups the elements of a sequence according to a specified key selector function. |

## Insert API:

|  |  |
| --- | --- |
| Method | Description |
| Insert | Inserts the specified record into the database table, and returns the number of rows affected. If the record has an auto-increment primary key, the method will update the object with the generated ID. |
| InsertOnly | Inserts the specified record into the database table, but only updates the object with the generated ID if it has an auto-increment primary key. Returns the number of rows affected. |
| InsertAll | Inserts multiple records into the database table in a single transaction, and returns the number of rows affected. |

## Update API:

|  |  |
| --- | --- |
| Method | Description |
| Update | Updates the specified record in the database table with the provided object, and returns the number of rows affected. |
| UpdateOnly | Updates only the specified properties of the record in the database table with the provided object, and returns the number of rows affected. |
| UpdateOnlyFields | Updates only the specified fields of the record in the database table with the provided object, and returns the number of rows affected. |
| Exists | Returns a boolean indicating whether a record with the specified primary key exists in the database table. |

## Delete API:

|  |  |
| --- | --- |
| Method | Description |
| Delete | Deletes the specified record from the database table, and returns the number of rows affected. |
| DeleteById | Deletes the record with the specified primary key from the database table, and returns the number of rows affected. |

## Sql utility API:

|  |  |
| --- | --- |
| Method | Description |
| Sql.In<T>(string fieldName, IEnumerable<T> values) | Creates a SQL IN clause for the specified field and values |
| Sql.NotIn<T>(string fieldName, IEnumerable<T> values) | Creates a SQL NOT IN clause for the specified field and values |
| Sql.Between<T>(string fieldName, T from, T to) | Creates a SQL BETWEEN clause for the specified field and range |
| Sql.Like(string fieldName, string value) | Creates a SQL LIKE clause for the specified field and value |
| Sql.Or(params SqlExpression[] expressions) | Creates a SQL OR clause for the specified expressions |
| Sql.And(params SqlExpression[] expressions) | Creates a SQL AND clause for the specified expressions |
| Sql.Exists<T>(SqlExpression<T> subQuery) | Creates a SQL EXISTS clause for the specified subquery |
| Sql.NotExists<T>(SqlExpression<T> subQuery) | Creates a SQL NOT EXISTS clause for the specified subquery |
| Sql.Count() | Creates a SQL COUNT function |
| Sql.Sum<T>(string fieldName) | Creates a SQL SUM function for the specified field |
| Sql.Avg<T>(string fieldName) | Creates a SQL AVG function for the specified field |
| Sql.Min<T>(string fieldName) | Creates a SQL MIN function for the specified field |
| Sql.Max<T>(string fieldName) | Creates a SQL MAX function for the specified field |

1. Dynamic Type

# Introduction

* In C#, the dynamic type is a type that is resolved at runtime instead of compile-time.
* This means that the type of the object is determined at runtime based on the value assigned to it.
* The dynamic type was introduced in C# 4.0 as a way to support dynamic binding, which allows developers to write code that interacts with objects whose type is not known until runtime.
* This is particularly useful when working with data from dynamic sources, such as web services, databases, or other external systems where the structure of the data may not be known until runtime.
* When a variable is declared as dynamic, the compiler does not perform any type checking on the variable, and any member access or method call on the variable is resolved at runtime based on the actual type of the object.
* This allows developers to write code that is more flexible and easier to maintain, as it can work with different types of objects without having to explicitly define each type.

**Syntax:**

*dynamic* <variable\_name> = value;

**Example:**

*namespace* DynamicTypeLearn

{

*public* *class* Program

    {

*public* *static* *void* Main(*string*[] args)

        {

*dynamic* x = 10;

            Console.WriteLine(x.GetType());

            x = "Dhruvil Dobariya";

            Console.WriteLine(x.GetType());

            x = Array.Empty<*int*>();

            Console.WriteLine(x.GetType());

        }

    }

}

Output:

System.Int32 System.String System.Int32[]

**Advantages:**

* Flexibility: dynamic allows developers to write code that can work with different types of objects without having to explicitly define each type.
* Interoperability: dynamic can be useful when working with data from dynamic sources, such as web services, databases, or other external systems where the structure of the data may not be known until runtime.
* Ease of use: dynamic can make code easier to write and read in some cases, as it can eliminate the need for casting or explicit type declarations.

**Drawbacks:**

* Performance: dynamic can introduce runtime overhead and slower performance compared to statically typed code, as the type is determined at runtime instead of compile-time.
* Debugging: dynamic can make debugging more difficult, as errors may not be caught until runtime and may be harder to trace back to the source.
* Safety: dynamic can introduce runtime errors if the actual type of the object does not have the member or method being accessed, which can be harder to catch and fix compared to compile-time errors.

1. Background Worker

# Introduction

* It is used to do time consuming task in background so our UI don’t block and all task work asynchronously.
* It provide abstract level of threading.
* Threading provide feature for run any task concurrently.
* Also thread provide more power to manage any task concurrently.
* But it’s also complex to manage.
* If we can’t manage appropriate way, then it’s raise issued like dead lock and synchronization related issues.
* Where background worker abstract level of threading, so here we have specific feature which help us to manage or run any time taken task concurrently.
* It provides less feature, but more easy to manage any task.
* It generally use for manage task that way so our UI can’t block.
* Where if we want to manage more complex task and synchronizations of tasks then we should use threading.

**Properties:**

|  |  |
| --- | --- |
| Property | Description |
| WorkerReportsProgress | Gets or sets a value indicating whether the background worker can report progress updates to the user interface thread. |
| WorkerSupportsCancellation | Gets or sets a value indicating whether the background worker supports cancellation. |
| CancellationPending | Gets a value indicating whether a cancellation request has been issued for the background operation. |
| IsBusy | Gets a value indicating whether the background worker is currently busy executing an operation. |
| RunWorkerCompletedEventArgs | Gets any results, error messages, or cancelled state from the completed background operation. |
| DoWorkEventArgs | Provides data for the DoWork event, which is raised when the background worker starts a new operation. |
| ProgressChangedEventArgs | Provides data for the ProgressChanged event, which is raised when the background worker reports progress during an operation. |

**Methods:**

|  |  |
| --- | --- |
| Method | Description |
| RunWorkerAsync | Starts the background operation. This method returns immediately and runs the DoWork event handler on a separate thread. |
| CancelAsync | Cancels the background operation if it supports cancellation. Raises the Cancelled event when the operation is cancelled. |
| ReportProgress | Reports progress updates from the background operation to the user interface thread. Raises the ProgressChanged event. |
| Dispose | Releases all resources used by the BackgroundWorker object. |
| OnDoWork | Raises the DoWork event. Override this method to add custom event handling logic. |
| OnRunWorkerCompleted | Raises the RunWorkerCompleted event. Override this method to add custom event handling logic. |
| OnProgressChanged | Raises the ProgressChanged event. Override this method to add custom event handling logic. |
| ReportProgressAsync | Reports progress updates from the background operation to the user interface thread asynchronously. Raises the ProgressChanged event. |