What is Delegate?

Delegate encapsulates method to treat it as first class object, when you declare delegate, a class is created on back end. Delegate can be added/deleted called chaining of delegate or multicast delegate. Through delegate, you can send method as parameter, for example:

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

public static class DeletageExample

{

private delegate void MyDelegate(string txt); *//Declare Delegate*

public static void DeletageExampleMethod()

{

MyDelegate del = SendText; *//Create Delegate object and assign function SendText*

del("Display this"); *//Invoke Delegate function*

}

public static void SendText(string txt)

{

Console.WriteLine(txt);

}

}

*//Delegate Chaining or Multicast Delegate*

public static class DeletageExample

{

private delegate void MyDelegate(string txt);

public static void DeletageExampleMethod()

{

MyDelegate txt1 = SendText;

MyDelegate txt2 = SendText2;

txt1 += txt2;

txt1("hay u");

}

public static void SendText(string txt)

{

Console.WriteLine(txt);

}

public static void SendText2(string txt)

{

Console.WriteLine(txt+" I am second delegate");

}

}

What is Anonymous Method?

A method without name, anonymous methods are used to shorten the delegates code, instead of creating one method and assigning it to delegate, method body can be written inside in delegate assigning statement, use anonymous method for only short methods alternative, e.g.

Hide   Copy Code

private delegate void MyDelegate(string txt);

public static void DeletageExampleMethod()

{

MyDelegate txt1 = delegate(string txt) { Console.WriteLine(txt); };

txt1("I am Anonymous method");

}

What is Lambda Expression?

Lambda expression is used to shorten the Anonymous methods, lambda operator(=>) is used in replacement of delegate keyword. e.g.

Hide   Copy Code

private delegate void MyDelegate(string txt);

public static void DeletageExampleMethod()

{

MyDelegate txt1 = txt => { Console.WriteLine(txt); }; *//Use () for more than*

*//one input parameters*

txt1("I am Anonymous method");

}

What is Generic Delegate?

In order to make delegate code shorter and remove delegate declaration, Action<>, Func<> generic delegates can be used.

* Action<>: No return type, can take up to 16 input parameters
* Func<>: Has return type, can take up to 16 input and one return type

Hide   Copy Code

public static void DeletageExampleMethod()

{

Action<string> txt1 = txt => Console.WriteLine(txt);

txt1("I don't have output ");

Func<string, string> txt2 = txt =>

{

Console.WriteLine(txt);

return "I am returning Value";

};

}

What is Event?

Events are kind of delegates who work in subscribe/Listener fashion. In other words, Event and Delegates work hand in hand. Following is a very basic example of Events about a person withdrawing money from bank. For any kind of account, money should be deducted from balance, Object Sender and EventArgs can be used to determine account type and request sender. To make the example simple, only basic functionality is implemented.

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

class Program

{

static void Main(string[] args)

{

var account1 = new AccountManagement();

var person = new Person(account1) { Name = "John Doe" };

account1.Winthdraw(person);

var account2 = new AccountManagement();

var person2 = new Person(account2) { Name = "Justin Phillip" };

account2.Winthdraw(person2);

Console.ReadLine();

}

}

public class Person

{

public string Name { get; set; }

private AccountManagement \_account;

public Person( AccountManagement account)

{

\_account = account;

account.Deposithandler += account.DeductMoney; *//Subscribe to Deduct Money Event*

}

}

public class AccountManagement

{

public delegate void DepostHandler(object sender, string eventArgs);

public event DepostHandler Deposithandler;

public void DeductMoney(object sender, string eventArgs)

{

Console.WriteLine(eventArgs + ": Money has been deducted from your account\n");

}

public void Winthdraw(Person person)

{

Console.WriteLine(person.Name+": You withdraw money");

Deposithandler(this, person.Name);

}

}

What is Implicit Typing or var Keyword?

Implicit typing through var keyword determines the data type of variable at the time of declaration and initialization. Var is different from Variant data type used in some languages and still holds strongly typed characteristics. Var variable cannot be initialize as null since the compiler cannot determine its data type (only Reference type variable can be declared as null). You must need to implicitly convert null value to some reference type, e.g., string.

Hide   Copy Code

var accountnum = "342425756342"; *//Correct, compiled as string*

accountnum = 50; *//Compiler error since it is considered as string in first statement.*

var username = null; *//Compiler error, unable to determine the datatype*

var username = (string)null; *//Successfully compile, string is reference.*

What is Code-First Approach in Entity Framework?

In Code-First approach, you do not worry about database design/development, you first create an application design, e.g., Model classes/entities, once you are done, you can create your class inherited from DbContext and create DBSet of your model classes. When you will access that DbContext class and perform some CRUD operation, the database will be automatically created by the name you gave in Context Constructor. Find that database in your local SQLExpress.

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

public class Student

{

public Student()

{

}

public int StudentID { get; set; }

public string StudentName { get; set; }

}

public class Standard

{

public Standard()

{

}

public int StandardId { get; set; }

public string StandardName { get; set; }

public string Description { get; set; }

}

public class Context : DbContext

{

public Context(): base("StudentDB.Test") *//Database Name*

{

}

public DbSet<Student> Students { get; set; }

public DbSet<Standard> Standards { get; set; }

}

class Program

{

static void Main(string[] args)

{

using (var ctx = new Context())

{

var stud = new Student() { StudentName = "New Student" };

ctx.Students.Add(stud);

ctx.SaveChanges();

}

}

}

What is Generics?

A simple definition is to separate the data from its type.There might be cases where you need to store or fetch the data in more than one type, e.g., you have student class, grade1 students ID is only integer and grade2students ID should be string, this is a stupid example but I am using it for proof of concepts. In order to efficiently implement class where you can validate integer and string IDs, you might need to declare two student classes but if you want to use one class for grade1 and grade2 students, generics can really help you.

Hide   Copy Code

static void Main(string[] args)

{

var Grad1Student = new Student<int> {StudentName = "John", StudentId = 1};

var Grad2Student = new Student<string> { StudentName = "John", StudentId = "R001" };

}

public class Student<T>

{

public string StudentName { get; set; }

public T StudentId { get; set; }

}

Following is an example where generic class object can be sent to method for further processing:

Hide   Copy Code

static void Main(string[] args)

{

var Grad1Student = new Student<int> {StudentName = "John", StudentId = 1};

var Grad2Student = new Student<string> { StudentName = "John", StudentId = "R001" };

var print = new Print();

print.PrintStudents(Grad1Student);

print.PrintStudents(Grad2Student);

Console.ReadLine();

}

public class Student<T>

{

public string StudentName { get; set; }

public T StudentId { get; set; }

}

public class Print

{

public void PrintStudents<T>(Student<T> student)

{

Console.WriteLine(student.StudentId);

}

}

What is Reflection?

Reflection is used to get meta data of class, object or assembly, e.g., assembly name and its version, list of all classes name in assembly, class's properties/fields and methods, etc. You can also get properties/fields values and invoke the methods. Reflection with Attribute can help to make very flexible application where you can access classes, methods and invoke them at run time, third party assemblies are easy to navigate through reflection and attribute. Following is a simple example.

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

static void Main(string[] args)

{

var assembies = Assembly.GetExecutingAssembly();

Console.WriteLine("assembies are " + assembies.FullName);

var classes = assembies.GetTypes();

foreach (var classname in classes)

{

Console.WriteLine("Classes are: " + classname.FullName);

foreach (var prop in classname.GetProperties())

{

Console.WriteLine("\t"+ prop.Name);

}

foreach (var method in classname.GetMethods())

{

Console.WriteLine("\t" + method.Name);

}

}

var onlyattrubuteclass =

assembies.GetTypes().Where(t => t.GetCustomAttributes<MyClassAttribute>().Any());

foreach (var attrubteclass in onlyattrubuteclass)

{

Console.WriteLine("\n\nAttributed Class " + attrubteclass.FullName);

}

Console.ReadLine();

}

[AttributeUsageAttribute(AttributeTargets.Class)]

public class MyClassAttribute : Attribute

{

}

[MyClass]

public class MyTest

{

public string Name { get; set; }

public int Id { get; set; }

public void TestMethod()

{

}

}

What is Extension Method?

Extension method as the name represents is used to extend the functionality of an existing class. This is a staticmethod having first parameter of class type needed to extend with extra keyword "this". Extension method is accessed like static class method with (.) operator but with object of class. Usually, Extension methods are good to use where we don't have code for third party DLL, but this perception varies from person to person. Following is a simple example of Extension Method.

Hide   Copy Code

class Program

{

static void Main(string[] args)

{

var test = new MyTest {Name = "John"};

test.MyTestExtesnionMethod();

Console.ReadLine();

}

}

public class MyTest

{

public string Name { get; set; }

public int Id { get; set; }

}

public static class ExtendMyTest

{

public static void MyTestExtesnionMethod(this MyTest test)

{

Console.WriteLine("I am extension method " + test.Name);

}

}

What is Dynamic and Late Binding?

Dynamic is a new type available in .NET to bypass compiler check during variable initialization and does it on run time. It is more flexible but dangerous and needs to be used more carefully. One cool feature of dynamic is to interact with other language code and execute them, e.g., in example PythonIron is used to interact with Python code and execute it. Also, dynamic can be used with ExpandoObject class available in .NET where you can declare properties and use them on run time. In ASP.NET MVC ViewBag is used to communicate data between Controller and Views which is a good example of dynamic. The simple example to run the Python code through dynamic is as follows: (Get IronPython from NuGet)

Hide   Copy Code

public static void Main()

{

var pythonRuntime = Python.CreateRuntime();

dynamic pythonFile = pythonRuntime.UseFile("Test.py");

pythonFile.SayHellotoPython();

Console.ReadLine();

}

Where *Python.py*is:

Hide   Copy Code

import sys;

def SayHellotoPython();

print "Hello I am Python"

ExpandoObject example is given below, it works like ViewBag in ASP.NET MVC:

Hide   Copy Code

public static void Main()

{

dynamic exp = new ExpandoObject();

exp.Name = "John";

exp.Age = 56;

Console.WriteLine(exp.Name);

Console.WriteLine(exp.Age);

Console.ReadLine();

}

What is an Optional Parameter?

Optional parameters are useful if you don't know or don't want to send value for all parameters every time you call method. It is a very good alternative of Method Overloading where you create a separate method for different set of parameters with the same name. There are a few things to remember when creating Optional Parameters, required parameters should be at the beginning and order of parameters really does matter, e.g., your method has 3 parameters, the first one is required and the other two are optional and you don't want to specify second parameter value, you will have compiler error because compiler would be unable to map the parameters, the workaround is using name parameter as given in the following example:

Hide   Copy Code

public static void Main()

{

MyTestExample("John");

MyTestExample("John", "Doe");

MyTestExample("John", "Doe",40);

MyTestExample(firstname:"John", age:67);

Console.ReadLine();

}

public static void MyTestExample(string firstname,string lastname=null, int age=0)

{

Console.WriteLine("{0} {1} is {2} years old", firstname, lastname, age);

}

What is TPL (Task Parallel Library)?

Task Parallel Library can be defined as an alternative of old Threading implementation and provides an easy way of implementing multiple tasks at same time. Task itself can be defined as basic execution process. Task(s) can be easily created by Task Factory and Task.WaitAll or WaitAny functions are helpful to establish hierarchy of Task or can be said as Task Blocking functions. e.g. halt execution until given task(s) functionality is not completed. Task can be continues means you can specify sub task if main task is completed. This functionality is very useful if you want to perform any functionality like logging, database saving, etc. after finishing main task execution. Following is a simple example of Task, ContinueWith and WaitAll.

Hide   Copy Code

static void Main(string[] args)

{

var task = Task.Factory.StartNew(() =>

MyFirtThread(1)).ContinueWith((prevtask) => ThreadCompleted("MyFirtThread"));

var task1 = Task.Factory.StartNew(() => MySecondThread(2));

var task2 = Task.Factory.StartNew(() => MyFirtThread(1));

var task3 = Task.Factory.StartNew(() => MySecondThread(2));

var tasklist = new List<Task> {task, task1, task2, task3};

Task.WaitAll(tasklist.ToArray());

Console.WriteLine("press enter to exit...");

Console.ReadLine();

}

public static void MyFirtThread(int id)

{

Thread.Sleep(1500);

Console.WriteLine("myFirtThread {0}",id);

}

public static void MySecondThread(int id)

{

Console.WriteLine("mySecondThread {0}",id);

}

public static void ThreadCompleted(string task)

{

Console.WriteLine(task + " completed");

}

What is Parallel Loop Available in Task Parallel Library?

There are ForEach and For loop available in Task Parallel Library that help to run the iteration parallel instead of sequential task execution as given in the first example that task is executed first, then task 1 to task 3. Parallel.For and Parallel.ForEach automatically handle WaitAll functionality that means it blocks the execution until all iterations of loop finish execution. Following is a simple example:

Hide   Copy Code

static void Main(string[] args)

{

var intArray = new List<int> {1, 2, 3, 4, 5, 6, 7, 8, 9, 23, 5, 7, 356, 89};

Parallel.ForEach(intArray, (i) => Console.WriteLine(i));

Console.WriteLine("press enter to exit...");

Console.ReadLine();

}

What is Task Cancellation Token or How to Cancel the Task?

Since Tasks execute parallel, there are very useful cancellation methods available. Through TaskCancellationToken, all tasks execution can be halted or handled (logging, skipping task, etc.) if some exception is raised. Following is a simple example of CancellationToken. CancellationTokenSourcegenerates the token to uniquely identify task, this token needs to be sent as parameter to task execution methods and check their property if Cancellation is requested or not, it is good practice to send Cancellation Token in all methods for better exception handling:

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

static void Main(string[] args)

{

var source = new CancellationTokenSource();

var task = Task.Factory.StartNew(() => MyFirtThread(1, source.Token)).ContinueWith

((prevtask) => ThreadCompleted("MyFirtThread", source.Token));

source.Cancel();

Console.WriteLine("press enter to exit...");

Console.ReadLine();

}

public static void MyFirtThread(int id, CancellationToken token)

{

if (token.IsCancellationRequested)

{

Console.WriteLine("Thread Cancellation Requested");

token.ThrowIfCancellationRequested();

}

Thread.Sleep(1500);

Console.WriteLine("myFirtThread {0}",id);

}

public static void ThreadCompleted(string task, CancellationToken token)

{

if (token.IsCancellationRequested)

{

Console.WriteLine("Thread Cancellation Requested");

token.ThrowIfCancellationRequested();

}

Console.WriteLine(task + " completed");

}

What is await and async?

await and async are constructs to work with Task Parallel Library, you can still achieve a lot of functionality through Task Parallel Library (TPL) classes, but await and async really save a lot of code and have built-in functionality that otherwise need more coding. async keyword is used in method signature, only tells that method should have await keyword to run it asynchronously, without await keyword, you will receive a warning from the compiler and the method will run synchronously. await executes code in non-blocking methodology. The good example of await and async is responsive UI, e.g. in desktop application, you cannot execute sub operation/thread if main thread is still in progress. If you click on any button, you are not able to do anything else until response is sent back from calling method. Through async and await, you can keep running sub thread even main thread is running that really increases the user experience. await and async are also helpful in many other scenarios, e.g., calling WebAPI, service or any web application, etc

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

PrintLabel("John Doe");

}

private async void PrintLabel(string name)

{

var result = await AssignValue(name);

label1.Text = result;

}

private Task<string> AssignValue(string name)

{

return Task.Factory.StartNew(() => GetValue(name));

}

private string GetValue(string name)

{

Thread.Sleep(2000);

return "Hi " + name;

}

}