Assembly : Unit of Deployment

.exe .dll

Console / Windows Templates > exe

Class Library > dll (dynamic link library)

Dll could be of two types

Private / Global

It is part of a project

Global Assembly > which is stored in GAC (common locations)

How to create Private Assembly :

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CommonLibrary

{

public class StringFunctions

{

public string CountForVowelsConsonants(string sentence)

{

StringBuilder sb = new StringBuilder();

return "ABC";

}

}

}

-------------------------------

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using CommonLibrary;

namespace Demo12thNov

{

class Program

{

static void Main(string[] args)

{

StringFunctions stringFunctions = new StringFunctions();

stringFunctions.CountForVowelsConsonants("This");

}

}

}

C:\Windows\Microsoft.NET\assembly\GAC\_MSIL

Abstract Class : A class in which some methods are defined and some are not defined(abstract methods)

The methods that are abstract need to be defined in Child class by using override keyword.

Purpose of Abstract Class : It is used to store common methods and it can be only inherited

It can not be instantiated as it’s is not complete

Interface : Syntactical contract in which we only declare methods, properties , indexers

It only contains methods / rules which has to be implemented by the classes which uses them

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Demo12thNov

{

interface IShape

{

void GetDetails();

void CalculatArea();

}

class A

{

}

interface I1

{

void Get();

void GetDetails();

}

class Rectangle : IShape, I1

{

void IShape.CalculatArea()

{

throw new NotImplementedException();

}

void I1.Get()

{

throw new NotImplementedException();

}

void IShape.GetDetails()

{

throw new NotImplementedException();

}

void I1.GetDetails()

{

throw new NotImplementedException();

}

public void Call()

{

}

}

class Square : A, IShape

{

int side, area;

public void CalculatArea()

{

area = side \* side;

}

public void GetDetails()

{

Console.WriteLine("Enter Side");

side = Convert.ToByte(Console.ReadLine());

}

public void DisplayArea()

{

Console.WriteLine("Area is " + area );

}

}

}

Polymorphism : One name , different forms

Static Polymorphism ,/ Compile Time Polymorphism > Methods are linked with objects at compile time

Run Time Polymorphism / Dynamic Polymorphism >Methods are linked with objects at run time

Static Polymorphism

static int add(int x, int y)

{

return x + y;

}

static int add (int x, int y, int z)

{

return x + y + z;

}

Dynamic : We achieve by using Virtual Functions

using System;

class Employee

{

int id;

string name;

public virtual void Get()

{

Console.Write("Enter ID");

id = int.Parse(Console.ReadLine());

Console.Write("Enter Name");

name = Console.ReadLine();

}

public virtual void Display()

{

Console.Write("ID is " + id);

Console.Write("Name is " + name);

}

public void salary()

{

Console.WriteLine("AA");

}

}

class PartTimeEmployee : Employee

{

string projectName;

public override void Get()

{

Console.Write("Enter projectName");

projectName = Console.ReadLine();

}

public override void Display()

{

Console.Write("projectName is " + projectName);

}

}

class FullTimeEmployee : Employee

{

string manager;

public override void Get()

{

Console.Write("Enter Manager");

manager = Console.ReadLine();

}

public override void Display()

{

Console.Write("Manager is " + manager);

}

}

class Program

{

static void Main()

{

Employee emp = new Employee();

emp.Get();

emp.Display();

Console.WriteLine("Parttime Class:");

PartTimeEmployee pt = new PartTimeEmployee();

emp = pt;

emp.salary();

emp.Get();

emp.Display();

}

}

using System;

abstract class Animals

{

abstract public void cats();

abstract public void dogs();

}

class Cats : Animals

{

public override void cats()

{

Console.WriteLine("Cats meow");

}

public override void dogs()

{

throw new NotImplementedException();

}

}

class Dogs : Animals

{

public override void cats()

{

}

public override void dogs()

{

Console.WriteLine("Dogs barks");

}

}

class CommonClass

{

public void GetAnimalType(Animals obj)

{

if (obj is Cats)

{

obj.cats();

}

else if (obj is Dogs)

{

obj.dogs();

}

}

}

class Program

{

static void Main(string[] args)

{

CommonClass commonClass = new CommonClass();

Animals animals = new Cats();

commonClass.GetAnimalType(animals);

//animals.cats();

animals = new Dogs();

animals.dogs();

}

}

Method Overriding : Redefining method of base class and child class

Class Base

{

Void Get()

}

Class child

{

void Get(){}

}

Base.Get();

Child.Get();

Abstract class

Base.Get();

Base.Get(); : Dynamic Poly

Sealed class is a class which can not be inherited

Generic : No type defined at compile time, type is known at run time ,its for every type

Generic collections: type are known at run time

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

ArrayList list = new ArrayList(); // We can only store in object form

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp29

{

class Str

{

String x;

public Str(string x)

{

this.x = x;

}

public void Write()

{

Console.WriteLine(x);

}

}

class Num

{

int x;

public Num(int x)

{

this.x = x;

}

public void Write()

{

Console.WriteLine(x);

}

}

class GenericClassDemo

{

public static void Main()

{

Num num = new Num(9);

num.Write();

}

}

}

Generic : Means where type is known at run time

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp29

{

class Num<T>

{

T x;

public Num(T x)

{

this.x = x;

}

public void Write()

{

Console.WriteLine(x);

}

}

class GenericClassDemo

{

public static void Main()

{

Num<int> num = new Num<int>(9);

num.Write();

Num<string> num1 = new Num<string>("AAAA");

num1.Write();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp29

{

class Num<T1, T2>

{

T1 x;

T1 y;

T1 res;

public Num(T1 x, T2 y)

{

dynamic x1 = x;

dynamic y1 = y;

this.res = x1 + y1;

}

public void Write()

{

Console.WriteLine(res);

}

}

class GenericClassDemo

{

public static void Main()

{

int x = 10;

//var y = 20;

//y = "aaa";

//dynamic z = "aaaa";

//z = 12;

//Num<int> num = new Num<int>(9);

//num.Write();

//Num<string> num1 = new Num<string>("AAAA");

//num1.Write();

Num<int, float> num1 = new Num<int, float>(10, 20);

}

}

}

Exception Handling

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp29

{

// Error : Unwanted output

// Syntax Error / Compile-Time Error : Easy to find out

// Logical Error / run-Time Error : Difficult to find out

// Exception : which might or might not occur

// We can not correct exception / cant avoid , handle them

// Try catch finally (optional)

class Exception\_Handling

{

static void Main()

{

int x, y;

try

{

Console.WriteLine("Enter Value of x");

x = Convert.ToByte(Console.ReadLine());

Console.WriteLine("Enter Value of y");

y = Convert.ToByte(Console.ReadLine());

int res = x / y;

Console.WriteLine("The result is " + res);

}

catch(Exception ex)

{

Console.WriteLine(ex.Message);

}

Console.WriteLine("Program ended");

}

}

}

Exception is the base class for all the Exceptions

This is for Handling exceptions of all types

We have specific Exception Handlers also

NullException

IndexOutOfRangeException

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp29

{

// Error : Unwanted output

// Syntax Error / Compile-Time Error : Easy to find out

// Logical Error / run-Time Error : Difficult to find out

// Exception : which might or might not occur

// We can not correct exception / cant avoid , handle them

// Try catch finally (optional)

class Exception\_Handling

{

static void Main()

{

int x, y;

try

{

Console.WriteLine("Enter Value of x");

x = Convert.ToByte(Console.ReadLine());

Console.WriteLine("Enter Value of y");

y = Convert.ToByte(Console.ReadLine());

int res = x / y;

Console.WriteLine("The result is " + res);

}

catch(DivideByZeroException ex)

{

Console.WriteLine("Number can not be divided by 0");

}

catch(Exception ex)

{

Console.WriteLine("SOME ERROR OCCURRED");

}

Console.WriteLine("Program ended");

}

}

}

Try catch finally

If there is some exception > Try >Catch > Finally

If there is some exception > Try > Finally

Finally block is used for clean up activities

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp29

{

// Error : Unwanted output

// Syntax Error / Compile-Time Error : Easy to find out

// Logical Error / run-Time Error : Difficult to find out

// Exception : which might or might not occur

// We can not correct exception / cant avoid , handle them

// Try catch finally (optional)

class Exception\_Handling

{

static void Main()

{

int x, y;

int[] num = new int[10];

try

{

Console.WriteLine("Enter Value of x");

x = Convert.ToByte(Console.ReadLine());

Console.WriteLine("Enter Value of y");

y = Convert.ToByte(Console.ReadLine());

int res = x / y;

num[11] = 90;

Console.WriteLine("The result is " + res);

}

catch(DivideByZeroException ex)

{

Console.WriteLine("Number can not be divided by 0");

}

catch(IndexOutOfRangeException ex)

{

Console.WriteLine("Index is out of the Range");

}

catch(Exception ex)

{

Console.WriteLine("SOME ERROR OCCURRED");

}

finally

{

Console.WriteLine("FINALLY BLOCK CALLED");

}

Console.WriteLine("Program ended");

}

}

}

RollNo shud not be more than 20

Length of name shud be min 10

Or this, we will create CustomException or UserDefined Exception

Create a class and inherit it from Exception class

Inside this create a constructor and pass the Message

Throw the exception when it occurs.

UserDefined Exception

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp29

{

class UserDefinedException : Exception

{

public string message;

public UserDefinedException(string Message)

{

this.message = Message;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp29

{

// Error : Unwanted output

// Syntax Error / Compile-Time Error : Easy to find out

// Logical Error / run-Time Error : Difficult to find out

// Exception : which might or might not occur

// We can not correct exception / cant avoid , handle them

// Try catch finally (optional)

class Exception\_Handling

{

static void Main()

{

int rn;

string name;

try

{

Console.WriteLine("Enter RollNo");

rn = Convert.ToByte(Console.ReadLine());

if(rn > 20)

{

throw new UserDefinedException("Roll No can not be more tahn 20");

}

Console.WriteLine("Enter Name");

name = Console.ReadLine();

if(name.Length < 10)

{

throw new UserDefinedException("Min 10 characters are needed");

}

Console.WriteLine("Roll No is " + rn);

Console.WriteLine("Name is " + name);

}

catch(UserDefinedException ex)

{

Console.WriteLine(ex.message);

}

catch(Exception ex)

{

Console.WriteLine("SOME ERROR OCCURRED");

}

finally

{

Console.WriteLine("FINALLY BLOCK CALLED");

}

Console.WriteLine("Program ended");

}

}

}

Struct : collection of data of different types

Purpose of struct : to store logically related data together

Struct is value type variable , It is also a user defined type

It does not support Inheritance

It has default constructor by default , but it does not allow to create default constructor explicitly

It only has one fully parameterized constructor

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp29

{

struct employee

{

int id;

string name;

// public Employee() { }

public employee(int id, string name)

{

this.id = id;

this.name = name;

}

public void Get() { }

public void Display() { }

}

class structDemo

{

public static void Main()

{

employee emp = new employee();

emp.Get();

emp.Display();

}

}

}

using System;

struct address

{

string houseno;

string city;

string state;

string pincode;

}

struct dob

{

int dd, mm, yy;

}

class Employee

{

int id;

string name;

address Address;

dob DOB;

public virtual void Get()

{

Console.Write("Enter ID");

id = int.Parse(Console.ReadLine());

Console.Write("Enter Name");

name = Console.ReadLine();

}

public virtual void Display()

{

Console.Write("ID is " + id);

Console.Write("Name is " + name);

}

public void salary()

{

Console.WriteLine("AA");

}

}

class PartTimeEmployee : Employee

{

string projectName;

public override void Get()

{

Console.Write("Enter projectName");

projectName = Console.ReadLine();

}

public override void Display()

{

Console.Write("projectName is " + projectName);

}

}

class FullTimeEmployee : Employee

{

string manager;

public override void Get()

{

Console.Write("Enter Manager");

manager = Console.ReadLine();

}

public override void Display()

{

Console.Write("Manager is " + manager);

}

}

class Program

{

static void Main()

{

Employee emp = new Employee();

emp.Get();

emp.Display();

Console.WriteLine("Parttime Class:");

PartTimeEmployee pt = new PartTimeEmployee();

emp = pt;

emp.salary();

emp.Get();

emp.Display();

emp = new Employee();

}

}

struct address

{

public string houseno;

public string city;

public string state;

public string pincode;

}

struct dob

{

public int dd, mm, yy;

}

class Employee

{

int id;

string name;

address Address = new address();

dob DOB = new dob();

public virtual void Get()

{

Console.Write("Enter ID");

id = int.Parse(Console.ReadLine());

Console.Write("Enter Name");

name = Console.ReadLine();

Console.WriteLine("ENter Address");

Address.city = Console.ReadLine();

}

------------

Enum is user defined type

What is Enum in C#?

 C# enum is a value type with a set of related named constants often referred as an enumerator list. The C# enum keyword is used to declare an enumeration. It is a primitive data type, which is user-defined. Enums type can be an integer (float, int, byte, double etc.) but if you use beside int, it has to be cast. Enum is used to create numeric constants in .NET framework. All member of the enum are of enum type. There must be a numeric value for each enum type.

The default underlying type of the enumeration elements is int. By default, the first enumerator has the value 0, and the value of each successive enumerator is increased by 1.

*enum Dow {Sat, Sun, Mon, Tue, Wed, Thu, Fri};*

***Program to demonstrate how to create and use an Enum***

1. **using** System;
2. **namespace** example\_enum {
3. **class** Program {
4. **public** **enum** DayofWeek {
5. Sunday = 1, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday
6. }
7. **static** **void** Main(**string**[] args) {
8. Console.WriteLine("Day of week {0} {1}", (**int**) DayofWeek.Sunday, DayofWeek.Sunday);
9. Console.WriteLine("Day of week {0} {1}", (**int**) DayofWeek.Monday, DayofWeek.Monday);
10. Console.WriteLine("Day of week {0} {1}", (**int**) DayofWeek.Tuesday, DayofWeek.Tuesday);
11. Console.WriteLine("Day of week {0} {1}", (**int**) DayofWeek.Wednesday, DayofWeek.Wednesday);
12. Console.WriteLine("Day of week {0} {1}", (**int**) DayofWeek.Thursday, DayofWeek.Thursday);
13. Console.WriteLine("Day of week {0} {1}", (**int**) DayofWeek.Friday, DayofWeek.Friday);
14. Console.WriteLine("Day of week {0} {1}", (**int**) DayofWeek.Saturday, DayofWeek.Saturday);
15. Console.ReadLine();
16. }
17. }
18. }

**Some points about enum**

* Enums are enumerated data type in C#.
* Enums are not for end-users, they are meant for developers.
* Enums are strongly typed constants. They are strongly typed, i.e., an enum of one type may not be implicitly assigned to an enum of another type even though the underlying value of their members are the same.
* Enumerations (enums) make your code more readable and understandable.
* enum values are fixed. enum can be displayed as a string and processed as an integer.
* The default type is int, and the approved types are byte, sbyte, short, ushort, uint, long, and ulong.
* Every enum type automatically derives from System.Enum and thus we can use System.Enum methods on enums.
* Enums are value types and are created on the stack and not on the heap.

**Can you give two same values in enum type?**

Yes, we can have the same value in the enum type. For example - when we want to set priority options like,

* Normal 0
* Excellent 1
* Default 0
* Good 3

**Program showing enum type having same values**

1. **using** System;
2. **namespace** enum\_example4 {
3. **class** Program {
4. **public** **enum** DayofWeek {
5. Sunday = 1, Monday, Tuesday = 1, Wednesday, Thursday = 2, Friday, Saturday
6. }
7. **static** **void** Main(**string**[] args) {
8. **string**[] values = Enum.GetNames(**typeof**(DayofWeek));
9. **foreach**(**string** s **in** values) {
10. Console.WriteLine(s);
11. }
12. Console.WriteLine();
13. **int**[] n = (**int**[]) Enum.GetValues(**typeof**(DayofWeek));
14. **foreach**(**int** x **in** n) {
15. Console.WriteLine(x);
16. }
17. Console.ReadLine();
18. }
19. }
20. }

**Program to find out the number of values in enum**

1. **using** System;
2. **namespace** enum\_exampl3 {
3. **class** Program {
4. **public** **enum** DayofWeek {
5. Sunday = 1, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday
6. }
7. **static** **void** Main(**string**[] args) {
8. **string**[] values = Enum.GetNames(**typeof**(DayofWeek));
9. **int** total = 0;
10. **foreach**(**string** s **in** values) {
11. Console.WriteLine(s);
12. total++;
13. }
14. Console.WriteLine("Total values in enum type is : {0}", total);
15. Console.WriteLine();
16. **int**[] n = (**int**[]) Enum.GetValues(**typeof**(DayofWeek));
17. **foreach**(**int** x **in** n) {
18. Console.WriteLine(x);
19. }
20. Console.ReadLine();
21. }
22. }
23. }
24. using System;
25. using System.Collections.Generic;
26. using System.Linq;
27. using System.Text;
28. using System.Threading.Tasks;
29. namespace ConsoleApp29
30. {
32. // It is used to assign constant names to a group of numeric integer values;
33. // It makes constants values more readbale.
34. // For ex. WeekDays.Monday makes it more readable than 0, ,where we are refferring to 0
35. // as Monday
36. enum Days { Monday , Tuesday, Wedneday , ThursDay, Friday };
37. class enumDemo
38. {
39. public static void Main()
40. {
41. Days days = Days.Monday;
42. Console.WriteLine((int)days);
43. int numDay = (int)Days.ThursDay;
44. Console.WriteLine(numDay);
45. }
46. }
47. }