Arrays

Collections

Generic Collections

Concepts of Oops

Class

Object

Constructors

Static Constructors

Def , Para, Copy , Private

Instance Variable / Static Variable

Static Functions (which can access only static variables)

Static Class

Static Methods can only access static members

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassDemo

{

class StaticDemo

{

static int A = 100;

static void add(int x, int y)

{

Console.WriteLine(x+y);

}

static void Main()

{

StaticDemo.A++;

add(10, 20);

}

}

}

Maths.sqrt(190);

Maths.Min(10,23,3);

**Static Class** : It’s a class in which all the members are static

**STATIC MEANS OBJECT NOT NEEDED . THAT PARTICULAR MEMBER CAN BE ACCESSEDD BY CLASS NAME**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassDemo

{

static class StringFunction

{

static string s1, s2;

static StringFunction()

{

s1 = string.Empty;

s2 = string.Empty;

}

public static int StringCompare( string s1, string s2)

{

if (s1.Length > s2.Length)

return 1;

else

return 0;

}

public static int StringCopy(string s1, string s2)

{

s1 = s2;

return 1;

}

}

class StaticDemo

{

static int A = 100;

static void add(int x, int y)

{

Console.WriteLine(x+y);

}

static void Main()

{

Console.WriteLine(StringFunction.StringCompare("This", "It");

;

Console.WriteLine("Enter");

StaticDemo.A++;

add(10, 20);

}

}

}

**Sealed Class : A class that can not be inherited**

sealed class Student

{

int rn;

string name;

string address;

public void GetDetails()

{

Console.WriteLine("Enter Roll No");

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

Console.WriteLine("Enter Name");

name = Console.ReadLine();

Console.WriteLine("Enter Address");

address = Console.ReadLine();

}

public void DisplayDetails()

{

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

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

Console.WriteLine("Address is " + address);

}

}

**class Sports : Student : Error will come here**

{

}

**Sealed Method > A method which cannot be overridden in child class**

**Abstract Class & Interface**

**1.In Abstract class , some of the methods are defined , some of the methods are not defined**

**In Interface , none of the method is defined**

**2. In Abstract class , we can declare variables**

**In Interface , we can not declare variables , but we can declare properties, indexers, methods , events**

**3. In Abstract Class, we have to use access specifier**

**In interface we do not use any access specifier, but they are public by default**

**Different types of Inheritance**

interface IA

{

void Get();

void show();

}

interface IB

{

void Get2();

void Show2();

}

class A : IA

{

public void Get()

{

Console.WriteLine("Inside Get");

}

public void show()

{

Console.WriteLine("Inside Show");

}

}

class A1 : IA

{

void IA.Get()

{

}

void IA.show()

{

}

}

**`**

**Named Para : What is its advantage**

**GetDetails(id:1, name :”Ajay”, Mark: 90);**

**There are ways to implement Interfaces**

**Implicit , explicit**

**In explicit interface names also comes alongwith the method names**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassDemo1

{

interface IA

{

void Get();

void show();

}

interface IB

{

void Get();

void Get2();

void Show2();

}

**// IMPLICIT WAY**

class C : IA, IB

{

public void Get()

{

throw new NotImplementedException();

}

public void Get2()

{

throw new NotImplementedException();

}

public void show()

{

throw new NotImplementedException();

}

public void Show2()

{

throw new NotImplementedException();

}

}

**// EXPLICIT WAY**

class D : IA, IB

{

void IA.Get()

{

throw new NotImplementedException();

}

void IB.Get()

{

throw new NotImplementedException();

}

void IB.Get2()

{

throw new NotImplementedException();

}

void IA.show()

{

throw new NotImplementedException();

}

public void Show2()

{

throw new NotImplementedException();

}

}

class A : IA

{

public void Get()

{

Console.WriteLine("Inside Get");

}

public void show()

{

Console.WriteLine("Inside Show");

}

}

class A1 : IA

{

void IA.Get()

{

}

void IA.show()

{

}

}

class InterfaceDemo2

{

}

}

**Multiple Inheritance is allowed thru Interface**

interface IA

{

void Get();

void show();

}

interface IB

{

void Get();

void Get2();

void Show2();

}

**class C : IA, IB // MULTIPLE INHERITANCE THRU INTERFACE**

{

}

**Enum**

**Is also a user defined type , it’s a value type variable**

An enumeration is a set of named integer constants. An enumerated type is declared using the **enum** keyword.

represents a group of **constants**

When to use enums : Whenever we have some limited options

Like Days Sunday , Monday ----------------- Saturday

1 – Sunday

2- Monday

Gender > Male Female Others

Colors : red Green yellow

Choice Add Subtract Multiple Divide

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassDemo

{

enum Days {Sunday=2 , Monday , Tuesday , Wednesday

, Thursday=0, Friday, Satudray};

class EnumDemo

{

static void Main()

{

Console.WriteLine((int)Days.Friday);

Console.WriteLine((int) Days.Monday);

}

}

}

**Without Using Enum**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassDemo

{

class EnumDemo

{

static void Main()

{

int x, y;

Console.WriteLine("Enter No1");

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

Console.WriteLine("Enter No2");

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

Console.WriteLine("Enter Choice");

int ch = byte.Parse(Console.ReadLine());

switch(ch)

{

case 1:

{

Console.WriteLine(x+y);

break;

}

case 2:

{

Console.WriteLine(x - y);

break;

}

case 3:

{

Console.WriteLine(x \* y);

break;

}

case 4:

{

Console.WriteLine(x / y);

break;

}

default:

{

Console.WriteLine("Invalid Choice");

break;

}

}

}

}

}

**Enums make program easy to understand for DEVELOPERS NOT END USERS**

**Using Enum**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassDemo

{

enum Days {Sunday=2 , Monday , Tuesday , Wednesday

, Thursday=0, Friday, Satudray};

enum Choice { Addition=1 , Subtraction,

Multiply, Divide };

class EnumDemo

{

static void Main()

{

int x, y;

Console.WriteLine("Enter No1");

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

Console.WriteLine("Enter No2");

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

Console.WriteLine("Enter Choice");

int ch = byte.Parse(Console.ReadLine());

switch(ch)

{

case (int)Choice.Addition:

{

Console.WriteLine(x+y);

break;

}

case (int)Choice.Subtraction:

{

Console.WriteLine(x - y);

break;

}

case (int)Choice.Multiply:

{

Console.WriteLine(x \* y);

break;

}

case (int)Choice.Divide:

{

Console.WriteLine(x / y);

break;

}

default:

{

Console.WriteLine("Invalid Choice");

break;

}

}

Console.WriteLine((int)Days.Friday);

Console.WriteLine((int) Days.Monday);

}

}

}

**Properties**

**Properties are wrappers around the variables**

**Set & get methods are used**

**Set is used to store value & get is used to return value**

class Student

{

int rn;

public int Rn

{

get

{

return rn;

}

set

{

rn = value;

}

}

}

**Properties shud be public, It allows a way to access private variables of a class**