ARRAY

using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;

namespace ArraysDemo  
{  
internal class Program  
{  
static void Main(string[] args)  
{  
//int a;  
//string str;  
int[] array1 = new int[5];  
string[] stringarray = new string[6];  
int[] array2 = new int[] { 1, 2, 3, 4 };

for(int i=0;i< array2.Length;i++)  
{  
Console.WriteLine(array2[i]);  
// Console.ReadLine();  
}

int[,] multidimetional = new int[2, 3];  
int[,] multidimentionalarray = { { 1, 2, 3 }, { 4, 5, 6 } };  
string[,] siblings = { { "Mike", "amy" }, { "mary", "albert" } };

foreach(string str in siblings)  
{  
Console.WriteLine("{0}",str);  
}  
//Console.ReadLine();

int[][] jagged\_arr = new int[4][];  
jagged\_arr[0]=new int[] {1,2,3,4};  
jagged\_arr[1] = new int[] { 11, 222, 33};  
jagged\_arr[2] = new int[] { 89,90 };  
jagged\_arr[3] = new int[] {0,4,5,6,7,8 };

for(int i=0;i< jagged\_arr.Length;i++)  
{  
Console.WriteLine(" Row ({0}) :", i);  
for(int j=0;j< jagged\_arr[i].Length;j++)  
{  
Console.WriteLine("{0}", jagged\_arr[i][j]);  
}  
}  
Console.ReadLine();

}  
}  
}

OUTREFERENCE

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Outreference

{

internal class Program

{

static void Main(string[] args)

{

int myvalue = 5;

Mymethod(out myvalue);

Console.WriteLine(myvalue);

Console.ReadLine();

}

static void Mymethod(out int param1)

{

param1 = 0;

param1 = param1+ 100;

}

}

}

OOPS CONCEPT PROGRAM WITH STATIC AND NON STATIC CLASS

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace OopsinCsharp

{

public class Employee

{

public Employee()

{

}

public static int staticMethod()

{

int x = 10;

int y = 20;

return x + y;

}

public string NonstaticMethod()

{

return "Hello GoodMorning";

}

}

internal class Program

{

static void Main(string[] args)

{

Employee emp=new Employee();

string str= emp.NonstaticMethod();

int intval= Employee.staticMethod();

Console.WriteLine(str);

Console.WriteLine(intval);

}

}

}

**INDEX CODE**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CsharpIndexers

{

public class Customer

{

List<Address> addresses=new List<Address>();

public Customer()

{

addresses.Add(new Address

{

Pincode = 2999033,

StreetName = "mumbai",

MobileNumber = "3434534"

});

addresses.Add(new Address

{

Pincode = 456456456,

StreetName = "Banglore",

MobileNumber = "6867867"

});

addresses.Add(new Address

{

Pincode = 4856456,

StreetName = "Hyderabad",

MobileNumber = "8945590"

});

}

public Address GetAddress(int pincode)

{

foreach (var item in addresses)

{

if(item.Pincode== pincode)

{

return item;

}

}

return null;

}

public Address GetAddress(string Mobilenumber)

{

foreach (var item in addresses)

{

if (item.MobileNumber == Mobilenumber)

{

return item;

}

}

return null;

}

public Address this[int pincode]

{

get

{

foreach (var item in addresses)

{

if (item.Pincode == pincode)

{

return item;

}

}

return null;

}

}

public Address this[string Mobilenumber]

{

get

{

foreach (var item in addresses)

{

if (item.MobileNumber == Mobilenumber)

{

return item;

}

}

return null;

}

}

public class Address

{

public int Pincode { get; set; }

public string StreetName { get; set; }

public string MobileNumber { get; set; }

}

}

public class Program

{

static void Main(string[] args)

{

Customer cname = new Customer();

var val= cname[2999033];

var val1 = cname["8945590"];

//cname[0] = 2999033;

////cname["8945590"];

//cname[] = 2999033;

// cname[2999033];

cname.GetAddress(2999033);

cname.GetAddress("8945590");

}

}

}

CUSTOM EXCEPTION CODE:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ChaepCustomException

{

public class TempIsZeroException : Exception

{

public TempIsZeroException(string message):base(message)

{

}

}

public class Temperature

{

int temparature = 0;

public void ShowTemp()

{

if(temparature == 0)

{

throw (new TempIsZeroException("Zero temprature found"));

}

else

{

Console.WriteLine("Temparature :{0}" ,temparature);

}

}

}

internal class Program

{

static void Main(string[] args)

{

Temperature temp=new Temperature();

try

{

temp.ShowTemp();

}

catch(TempIsZeroException e)

{

Console.WriteLine(" Tempzeroexception :{0}",e.Message);

}

}

}

}

IE NUMERABLE CODE

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CsharpIenumarable

{

internal class Program

{

static void Main(string[] args)

{

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

oyears.Add(1990);

oyears.Add(1991);

oyears.Add(1992);

oyears.Add(1993);

oyears.Add(2000);

oyears.Add(2001);

oyears.Add(2002);

IEnumerable<int> ienum =(IEnumerable<int>) oyears;

foreach (var i in ienum)

{

Console.WriteLine(i);

}

IEnumerator<int> ienumarator = oyears.GetEnumerator();

Console.WriteLine("IEnumarator values");

while (ienumarator.MoveNext())

{

Console.WriteLine(ienumarator.Current.ToString());

}

Console.ReadLine();

}

}

}

IEnumerator vs IEnumerable

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CsharpIenumarable

{

internal class Program

{

static void Main(string[] args)

{

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

oyears.Add(1990);

oyears.Add(1991);

oyears.Add(1992);

oyears.Add(1993);

oyears.Add(2000);

oyears.Add(2001);

oyears.Add(2002);

IEnumerable<int> ienum =(IEnumerable<int>) oyears;

foreach (var i in ienum)

{

Console.WriteLine(i);

}

IEnumerator<int> ienumarator = oyears.GetEnumerator();

//Console.WriteLine("IEnumarator values");

//while (ienumarator.MoveNext())

//{

// Console.WriteLine(ienumarator.Current.ToString());

//}

Iterate19to2000(ienumarator);

Console.ReadLine();

}

static void Iterate19to2000(IEnumerator<int> o)

{

Console.WriteLine("From 1990 to 2000");

while(o.MoveNext())

{

Console.WriteLine(o.Current.ToString());

if (Convert.ToInt32(o.Current.ToString())>=2000)

{

Iteratefrom2000andabove(o);

}

}

}

static void Iteratefrom2000andabove(IEnumerator<int> o)

{

Console.WriteLine("from 2001 and above");

while (o.MoveNext())

{

Console.WriteLine(o.Current.ToString());

}

}

}

}

GENERIC METHOD CODE:

using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;

namespace CsharpGenericClass  
{

class GenericClass<T>  
{  
public GenericClass(T msg)  
{  
Console.WriteLine(msg);  
}  
  
}  
internal class Program  
{  
static void Main(string[] args)  
{  
GenericClass<string> gen = new GenericClass<string>("this is generic class");  
GenericClass<int> geni = new GenericClass<int>(101);  
GenericClass<char> genc = new GenericClass<char>('a');

}  
}  
}

Generic constraints code

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CsharpGenericConstraint

{

public class GenericClass<T> where T : class

{

public T msg;

public void GenericMethod(T name,T location)

{

Console.WriteLine("{0}", msg);

Console.WriteLine(" Name :{0}", name);

Console.WriteLine("{0}", location);

}

}

internal class Program

{

static void Main(string[] args)

{

GenericClass<string> gclass = new GenericClass<string>();

GenericClass<object> gclass1 = new GenericClass<object>();

//GenericClass<int> gclass2 = new GenericClass<int>();

}

}

}

MULTICAST DELEGATES CODE

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CsharpMulticastDelegate

{

public delegate void MyDelegate(string msg);

internal class Program

{

static void Main(string[] args)

{

MyDelegate del1 = ClassA.MethodA;

MyDelegate del2 = ClassB.MethodB;

MyDelegate del = del1 + del2;

Console.WriteLine("After del1+Del2");

del("Hello world");

MyDelegate del3 = (string msg) => Console.WriteLine("called lamda expresssion "+ msg);

del += del3;

Console.WriteLine("after del1,del2,del3");

del("Good morning");

del = del - del2;

Console.WriteLine("after del-del2");

del("Afternoon");

del -= del1;

Console.WriteLine("after delteting del1 and del2");

del("Evening");

}

}

public class ClassA

{

public static void MethodA(string message)

{

Console.WriteLine(" Called Class A.Method A with parameter :"+ message);

}

}

public class ClassB

{

public static void MethodB(string message)

{

Console.WriteLine(" Called Class B.Method B with parameter :" + message);

}

}

}

ANONYMOUS Method example 2

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Csharpanomanousmethodmethod1

{

delegate bool IsTeenager(Student stud);

internal class Program

{

static void Main(string[] args)

{

IsTeenager Isteenagerval = delegate (Student s)

{

return s.Age > 12 && s.Age < 20;

};

Student stud = new Student() { Age = 18 };

Console.WriteLine(Isteenagerval(stud));

Console.ReadLine();

}

}

public class Student

{

public int Id { get; set; }

public string Name { get; set; }

public int Age { get; set; }

}

}

Reflection code

using System;

using System.Reflection;

namespace CsharpReflection

{

class Program

{

static void Main(string[] args)

{

string path = @"C:\Users\nimai\source\repos\ConsoleApp8\ConsoleApp8\bin\Debug\netcoreapp3.1\ConsoleApp8.dll";

Assembly assembly = Assembly.LoadFile(path);

Type[] types = assembly.GetTypes();

foreach (var type in types)

{

Console.WriteLine(" Class Name :" + type.Name);

MethodInfo[] methods = type.GetMethods();

foreach (var method in methods)

{

Console.WriteLine("method name :" + method.Name);

ParameterInfo[] parameters = method.GetParameters();

foreach (var paramvals in parameters)

{

Console.WriteLine("parameter values :" + paramvals.Name);

}

}

}

Console.ReadLine();

}

}

}

PARALLEL PROGRAMMING CODE

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Threading;

namespace CsharpTaskParallelProgrammingdemo

{

internal class Program

{

static void Main(string[] args)

{

Console.WriteLine("Task parallel Foreach loop");

DateTime startDatetime=DateTime.Now;

Console.WriteLine($"foreach loop start at={startDatetime}");

List<int> intergerlist=Enumerable.Range(0,10).ToList();

foreach(int i in intergerlist)

{

long total = DosomeIndepedentTimeConsumingTask();

Console.WriteLine($"{i}-{total}");

}

DateTime endDatetime=DateTime.Now;

Console.WriteLine($"foreach loop end at :{endDatetime}");

TimeSpan span = endDatetime - startDatetime;

int ms =(int) span.TotalMilliseconds;

Console.WriteLine($"Time taken by foreach loop in miliseconds {ms}");

Console.WriteLine("Task parallel Foreach loop");

DateTime startDatetime1 = DateTime.Now;

Console.WriteLine($"foreach loop start at={startDatetime1}");

List<int> intergerlist1 = Enumerable.Range(0, 10).ToList();

Parallel.ForEach(intergerlist1,i =>

{

long total = DosomeIndepedentTimeConsumingTask();

Console.WriteLine($"{i}-{total}");

});

DateTime endDatetime1 = DateTime.Now;

Console.WriteLine($"foreach loop end at :{endDatetime1}");

TimeSpan span1 = endDatetime1 - startDatetime1;

int ms1 = (int)span1.TotalMilliseconds;

Console.WriteLine($"Time taken by foreach loop in miliseconds {ms1}");

Console.ReadLine();

}

static long DosomeIndepedentTimeConsumingTask()

{

long total = 0;

for(int i=1;i<100000000;i++)

{

total += i;

}

return total;

}

}

}

PATTERN Matching

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CSharp7Patternmatching

{

internal class Program

{

static void Main(string[] args)

{

Student stud=new Student("Kamalesh","Ram");

switch(stud)

{

case null:

Console.WriteLine("its constant pattern");

break;

case Student s when s.Firstname.ToLower().StartsWith("k".ToLower())

: Console.WriteLine(s.Firstname);

break;

case var x:

Console.WriteLine(x.GetType().Name);

break;

}

}

}

public class Student

{

public Student(string fir,string lastval)

{

}

public string Firstname { get; set; }

public string LastName { get; set; }

}

}