

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace IInterface

{

public enum Acess : byte

{

Director = 1,

Programmer = 2,

Manager = 3

}

public interface IEmployee:IComparable

{

void ShowPersone();

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using IInterface;

namespace AttributesAccessLibaryEmpl

{

[AttributeUsage(AttributeTargets.Class)]

public class AccessLevelAttribute:Attribute

{

public Acess Acess { get; set; }

private **DateTime** dt;

public string Dt

{

set { dt = Convert.ToDateTime(value); }

get { return dt.ToShortDateString(); }

}

public AccessLevelAttribute(Acess acess)

{

this.Acess = acess;

}

}

}

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using IInterface;

namespace AttributesAccessLibaryEmpl

{

public abstract class Employee : IEmployee

{

protected string Name;

public abstract void ShowPersone();

public int CompareTo(object obj)

{

return Name.CompareTo(((Employee)obj).Name);

}

}

[AccessLevel(Acess.Manager, Dt = "28/11/2014")]

public class Menedger : Employee

{

public Menedger(string name)

{

Name = name;

}

public override void ShowPersone()

{

Acess acc =

typeof (Menedger).GetCustomAttributes(typeof (AccessLevelAttribute), false)

.OfType<AccessLevelAttribute>()

.Select(n=>n.Acess).SingleOrDefault();

Console.WriteLine("Name - {0,-10}, Acess - {1,-10}",Name,acc);

}

}

[AccessLevel(Acess.Programmer, Dt = "20/11/2014")]

public class Programmer : Employee

{

public Programmer(string name)

{

Name = name;

}

public override void ShowPersone()

{

Acess acc =

typeof(Menedger).GetCustomAttributes(typeof(AccessLevelAttribute), false)

.OfType<AccessLevelAttribute>()

.Select(n => n.Acess).SingleOrDefault();

Console.WriteLine("Name - {0,-10}, Acess - {1,-10}", Name, acc);

}

}

[AccessLevel(Acess.Director, Dt = "30/11/2014")]

public class Director : Employee

{

public Director(string name)

{

Name = name;

}

public override void ShowPersone()

{

Acess acc =

typeof(Menedger).GetCustomAttributes(typeof(AccessLevelAttribute), false)

.OfType<AccessLevelAttribute>()

.Select(n => n.Acess).SingleOrDefault();

Console.WriteLine("Name - {0,-10}, Acess - {1,-10}", Name, acc);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading.Tasks;

using IInterface;

namespace AttributesEmployee

{

class Program

{

static void Main(string[] args)

{

List<IEmployee> list = new List<IEmployee>

{

Employee("AttributesAccessLibaryEmpl.Programmer", "Sanek"),

Employee("AttributesAccessLibaryEmpl.Director", "Victor"),

Employee("AttributesAccessLibaryEmpl.Menedger", "Ivan")

};

list.Sort();

IsSuccesEnterRoom(list);

Console.ReadKey();

}

private static IEmployee Employee(string str,string name)

{

Assembly ass = Assembly.LoadFile(@"C:\Users\Виктор\Documents\Visual Studio 2013\Projects\HomeWork\AttributesAccessLibaryEmpl\obj\Debug\AttributesAccessLibaryEmpl.dll");

Type type = ass.GetType(str);

ConstructorInfo constructor = type.GetConstructor(new Type[] {typeof (string)});

IEmployee m = constructor.Invoke(new object[] { name }) as IEmployee;

return m;

}

private static void IsSuccesEnterRoom(IEnumerable<IEmployee> emp)

{

emp.ToList().ForEach(n=>n.ShowPersone());

var list1 = from d in emp

select new

{

acc = ((dynamic)d.GetType().GetCustomAttributes(false).First()).Acess,

dt = ((dynamic)d.GetType().GetCustomAttributes(false).First()).Dt

};

var list = from d in emp

from dynamic dy in d.GetType().GetCustomAttributes(false)

select new

{

acc=dy.Acess,

dt=dy.Dt

};

foreach (var item in list1)

{

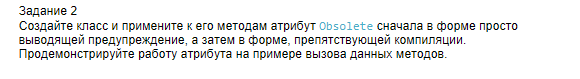
Console.WriteLine(item.acc +" - "+item.dt);

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading.Tasks;

namespace ObsoletAttribute

{

class MyClass

{

[Obsolete("Old method",true)]

public void MyMethod()

{

Console.WriteLine("MyMethod");

}

}

class Program

{

static void Main(string[] args)

{

MyClass m=new MyClass();

m.MyMethod();

MethodInfo mi = m.GetType().GetMethod("MyMethod");

ObsoleteAttribute methodAttributes = mi.GetCustomAttribute<ObsoleteAttribute>(false);

Console.WriteLine("{0,-10}{1,10}",methodAttributes.Message,methodAttributes.IsError);

Console.ReadKey();

}

}

}