

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HomeWork

{

class GetMounth

{

public Mounth GetDays(ICollection<Mounth> coll, int days)

{

foreach (var item in coll)

{

if (item.Days == days)

return item;

}

throw new KeyNotFoundException("Not found mounth by days");

}

public Mounth GetMount(ICollection<Mounth> coll,int pos)

{

foreach (var item in coll)

{

if (item.Count == pos)

return item;

}

throw new KeyNotFoundException("Not found mounth by pos");

}

}

class MyCollection<T>:ICollection<T>,IEnumerator<T> where T:class, new()

{

T [] array=new T[0];

private int \_pos;

public MyCollection()

{

\_pos = -1;

}

public IEnumerator<T> GetEnumerator()

{

return this;

}

IEnumerator IEnumerable.GetEnumerator()

{

return (this as IEnumerable<T>).GetEnumerator();

}

public void Add(T item)

{

T[] tmp=new T[Count+1];

array.CopyTo(tmp,0);

tmp[Count] = item;

array = tmp;

}

public void Clear()

{

array=new T[12];

}

public bool Contains(T item)

{

return array.Contains(item);

}

public void CopyTo(T[] array, int arrayIndex)

{

this.array.CopyTo(array,arrayIndex);

}

public bool Remove(T item)

{

return false;

}

public int Count {

get { return array.Length; }

}

public bool IsReadOnly {

get { return false; }

}

public void Dispose()

{

Reset();

}

public bool MoveNext()

{

if (\_pos + 1 < Count)

{

\_pos++;

return true;

}

return false;

}

public void Reset()

{

\_pos = -1;

}

public T Current {

get { return array[\_pos]; }

}

object IEnumerator.Current

{

get { return Current; }

}

}

}

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HomeWork

{

class Mounth

{

public int Days { get; set; }

public int Count { get; set; }

public override string ToString()

{

return Days + " - " + Count;

}

}

class Program

{

static void Main(string[] args)

{

int[] mounth = {31,28,31,30,31,30,31,31,30,31,30,31};

MyCollection<Mounth> collection = new MyCollection<Mounth>();

for (int i = 0; i < mounth.Length; i++)

{

collection.Add(new Mounth{Count = i,Days = mounth[i]});

}

foreach (var mounth1 in collection)

{

Console.WriteLine("{0} - days = {1}", mounth1.Count, mounth1.Days);

}

try

{

Console.WriteLine(collection.Contains(new Mounth()));

Console.WriteLine(new GetMounth().GetMount(collection,11));

}

catch (Exception e)

{

Console.WriteLine(e.Message);

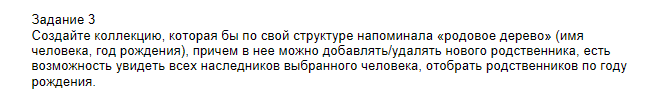
}

Console.ReadKey();

}

}

}



using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HomeWork

{

interface IFamaly

{

string Name { get; set; }

IFamaly Parent { get; set; }

}

class FamalyCollection:IFamaly

{

public string Name { get; set; }

public IFamaly Parent { get; set; }

public FamalyCollection(string name, IFamaly parent)

{

Name = name;

Parent = parent;

}

static public IFamaly CreateFamaly(IEnumerable arr)

{

IEnumerator inum = arr.GetEnumerator();

return CreateFamaly(inum);

}

static public IFamaly CreateFamaly(IEnumerator inum)

{

IFamaly fam = null;

if (inum.MoveNext())

{

return fam = new FamalyCollection(inum.Current.ToString(), CreateFamaly(inum));

}

return null;

}

}

}

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HomeWork

{

static class ExtensionMethod

{

static public IEnumerable ShowFamaly(this IFamaly f)

{

for (IFamaly i = f; i.Parent != null; i=i.Parent)

{

yield return "Name - " + i.Name + "; parent - " + i.Parent.Name;

}

}

}

class Program

{

static void Main(string[] args)

{

string[] str = {"Vitek", "Sanek", "Ivan"};

IFamaly coll = FamalyCollection.CreateFamaly(str);

foreach (var VARIABLE in coll.ShowFamaly())

{

Console.WriteLine(VARIABLE);

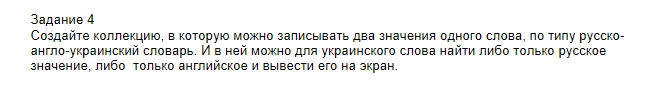
}

Console.ReadKey();

}

}

}



using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HomeWork

{

interface IWorld

{

string Ukr { get; set; }

string Rus { get; set; }

string Eng { get; set; }

string Contains(string item);

}

class Worlds:IWorld

{

public string Ukr { get; set; }

public string Rus { get; set; }

public string Eng { get; set; }

public override string ToString()

{

return Rus + " - " + Eng+" "+Ukr;

}

public string Contains(string item)

{

if (Ukr.Equals(item))

return Rus+" - "+Eng;

if (Rus.Equals(item))

return Ukr + " " + Eng;

if (Eng.Equals(item))

return Ukr+" "+Rus;

return null;

}

}

class WorldsCollection

{

private IWorld[] list = new IWorld[2];

public string this[String v]

{

get { return GetWorld(v); }

}

public WorldsCollection()

{

list[0]=new Worlds{Eng = "book",Rus="книга",Ukr = "пидручник"};

list[1] = new Worlds { Eng = "house", Rus = "дом", Ukr = "будинок" };

}

public string GetWorld(string v)

{

for (int i = 0; i < list.Length; i++)

{

if (list[i].Contains(v) != null)

{

return list[i].Contains(v);

}

}

return null;

}

public IEnumerator<IWorld> GetEnumerator()

{

for (int i = 0; i < list.Length; i++)

{

yield return list[i];

}

}

}

}

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HomeWork

{

class Program

{

static void Main(string[] args)

{

WorldsCollection wc=new WorldsCollection();

IEnumerator<IWorld> iw = wc.GetEnumerator();

foreach (var VARIABLE in wc)

{

Console.WriteLine(VARIABLE.Eng);

}

while (iw.MoveNext())

{

Console.WriteLine(iw.Current);

}

Console.WriteLine(wc["book"]);

Console.ReadKey();

}

}

}