**Создайте расширяющий метод: public static T[] GetArray(this MyList list) Примените расширяющий метод к экземпляру типа MyList. Выведите на экран значения элементов массива, который вернул расширяющий метод GetArray().**

using System;

using System.Collections;

using System.Collections.Generic;

namespace Lab2

{

class Program

{

static void Main(string[] args)

{

MyList<char> mylist = new MyList<char>();

for (int i = 15; i < 20; i++)

{

mylist.Add((char)i);

}

Console.WriteLine(new string('=', 15));

Console.WriteLine(mylist.Count);

Console.WriteLine(new string('=', 15));

// IEnumerable<char> enumerable = mylist as IEnumerable<char>;

for (int i = 0; i < mylist.Count; i++)

{

Console.WriteLine(mylist.GetArray()[i]);

}

}

}

public static class TClass

{

public static T[] GetArray<T>(this IEnumerable<T> list)

{

int i = 0; T[] array = new T[i];

foreach (T item in list)

{

T[] NewArray = new T[array.Length + 1];

array.CopyTo(NewArray, 0);

NewArray[array.Length] = item;

array = NewArray;

}

return array;

}

}

class MyList<T> : IEnumerable<T>

{

T[] array = new T[0];

public T this[int index]

{

get

{

return array[index];

}

}

public void Add(T item)

{

T[] NewArray = new T[array.Length + 1];

array.CopyTo(NewArray, 0);

NewArray[array.Length] = item;

array = NewArray;

}

public int Count

{

get { return array.Length; }

}

int position = -1;

public object Current

{

get { return array[position]; }

}

public bool MoveNext()

{

if (position < array.Length - 1)

{

position++;

return true;

}

else { Reset(); return false; }

}

public void Reset()

{

position = -1;

}

public IEnumerator<T> GetEnumerator()

{

foreach (T item in array)

yield return item;

}

IEnumerator IEnumerable.GetEnumerator()

{

return array.GetEnumerator();

}

}

}

**Создайте класс MyList. Реализуем в простейших приближении возможность использования его экземпляра аналогично экземпляр класса List. Минимально Требуемый интерфейс взаимодействия с экземпляром, должен включать метод добавления элемента, Индексатором для получения значения элемента по указанному индексу и свойство только для чтения для получения общего количества элементов.**

using System;

namespace Lab2

{

class MyList<T>

{

private T[] myList = null;

public T this[int index]

{

get { return myList[index]; }

set { myList[index] = value; }

}

public MyList()

{

this.myList = new T[1];

}

public MyList(int count)

{

this.myList = new T[count];

}

public void Add(T item)

{

T[] extendedList = new T[myList.Length + 1];

extendedList[extendedList.Length - 1] = item;

myList = extendedList;

}

public int Capacity

{

get { return myList.Length; }

}

// Возвращает число элементов, которые фактически содержатся в коллекции MyList<T>.

public int Count

{

get

{

int count = 0;

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

{

if (myList[i].ToString() != null)

{

count++;

}

}

return count;

}

}

}

class Program

{

static void Main(string[] args)

{

MyList<int> mylist = new MyList<int>(5);

mylist[0] = 1;

Console.WriteLine("Емкость списка: {0} элемент(-ов)", mylist.Capacity);

Console.WriteLine("Список фактически содержит: {0} элемент(-ов)", mylist.Count);

mylist.Add(15);

Console.WriteLine("Емкость списка: {0} элемент(-ов)", mylist.Capacity);

Console.ReadKey();

}

}

}

**Создайте класс MyClass, содержащий статический фабричный метод – T FacrotyMethod(), который будет порождать экземпляры типа, указанного в качестве параметра типа (указателя места заполнения типом – Т).**

using System;

namespace Lab2

{

class Program

{

static void Main(string[] args)

{

var x = MyClass<bool>.FactoryMethod();

var y = MyClass<decimal>.FactoryMethod();

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

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

}

}

class MyClass<T>

where T : new()

{

public static T FactoryMethod()

{

return new T();

}

}

}

**Создайте класс MyDictionary. Реализуйте в простейшем приближении возможность использования его экземпляра аналогично экземпляру класса Dictionary. Минимально требуемый интерфейс взаимодействия с экземпляром, должен включать метод добавления пар элементов, индексатор для получения значения элемента по указанному индексу и свойство только для чтения для получения общего количества пар элементов.**

using System;

using System.Collections;

using System.Collections.Generic;

namespace Lab2

{

class MyDictionary<Tkey, Tval>

{

private int counter = 0;

private Tkey[] keysArray = null;

private Tval[] valsArray = null;

public int Counter

{

get { return this.counter; }

}

public void Add(Tkey key, Tval val)

{

this.counter++;

Array.Resize(ref keysArray, counter);

keysArray[counter - 1] = key;

Array.Resize(ref valsArray, counter);

valsArray[counter - 1] = val;

}

public Tval this[Tkey key]

{

get

{

int ind = 0;

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

{

if (key.Equals(keysArray[i]))

ind = i;

}

return valsArray[ind];

}

}

}

}

Начало формы

Конец формы