

Лабораторная работа № 13

Разработка программы, управляемой событиями

Цель. Получить практические навыки разработки программы, управляемой событиями, использования делегатов и событий..

Постановка задачи

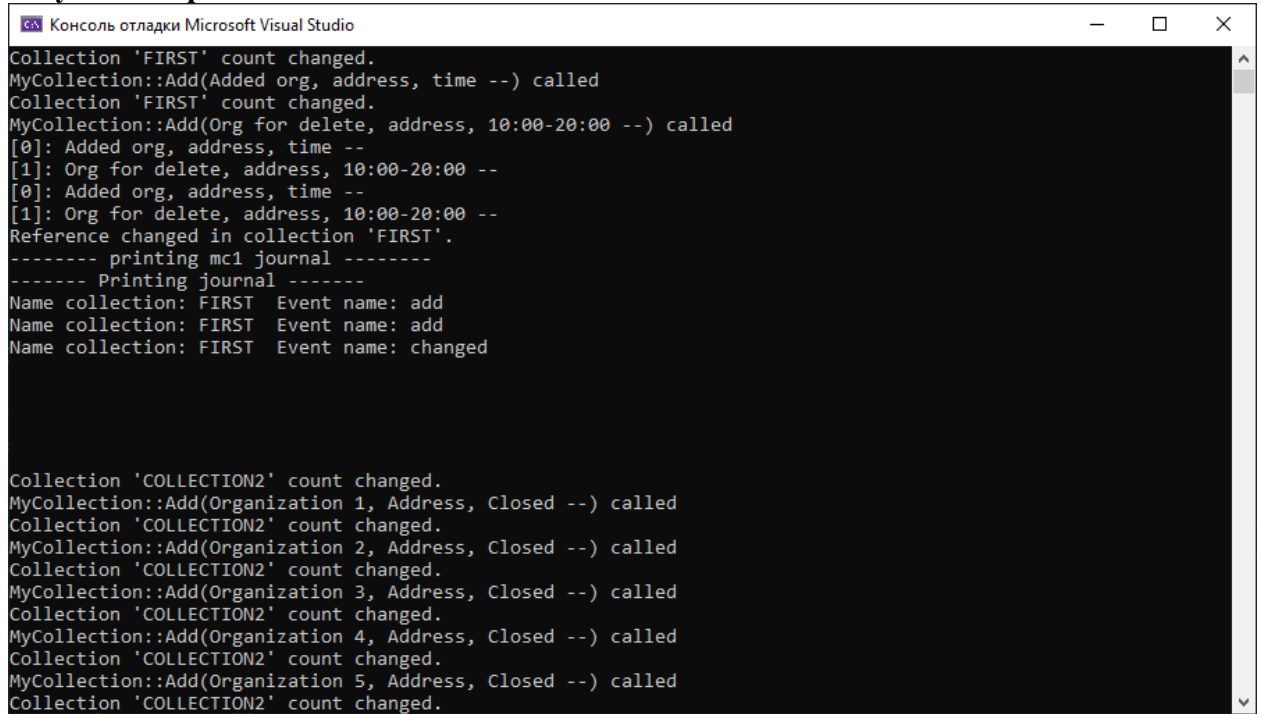
1. Создать иерархию классов (см. лаб. 10). Для каждого класса реализовать конструктор без параметров, с параметрами, свойства для доступа к полям объектов, метод для автоматического формирования объектов. Перегрузить метод ToString() для формирования строки со значениями всех полей класса.
2. Создать класс MyCollection как производный класс от класса Collection<MyClass>.
3. Класс Collection<MyClass> взять из лабораторной работы №12. В классе должны быть реализованы в классе методы для заполнения коллекции (элементы коллекции формируются автоматически), добавления элементов коллекции, удаления элементов коллекции, сортировки элементов коллекции по заданному полю, очистки коллекции, реализован итератор для доступа к элементам коллекции, реализовано свойство Length (только для чтения), содержащее текущее количество элементов коллекции.
4. Определить класс MyNewCollection производный от класса MyCollection, который с помощью событий извещает об изменениях в коллекции. Коллекция изменяется:
 - при удалении/добавлении элементов
 - при изменении одной из входящих в коллекцию ссылок, например, когда одной из ссылок присваивается новое значение.В этом случае в соответствующих методах или свойствах класса бросаются события.
5. В новую версию класса MyNewCollection добавить
 - открытое автореализуемое свойство типа string с названием коллекции;
 - метод bool Remove (int j) для удаления элемента с номером j ; если в списке нет элемента с номером j, метод возвращает значение false;
 - индексатор (с методами get и set) с целочисленным индексом для доступа к элементу с заданным номером.
6. Для событий, извещающих об изменениях в коллекции, определяется свой делегат CollectionHandler с сигнатурой:
void CollectionHandler (object source, CollectionHandlerEventArgs args);
7. Для передачи информации о событии определить класс CollectionHandlerEventArgs, производный от класса System.EventArgs, который содержит
 - открытое автореализуемое свойство типа string с названием коллекции, в которой произошло событие;
 - открытое автореализуемое свойство типа string с информацией о типе изменений в коллекции;
 - открытое автореализуемое свойство для ссылки на объект, с которым связаны изменения;
 - конструкторы для инициализации класса;
 - перегруженную версию метода string ToString() для формирования строки с информацией обо всех полях класса.
8. В класс MyNewCollection добавить два события типа CollectionHandler.
 - CollectionCountChanged, которое происходит при добавлении нового элемента в коллекцию или при удалении элемента из коллекции; через объект CollectionHandlerEventArgs событие передает имя коллекции, строку с информацией о том, что в коллекцию был добавлен новый элемент или из нее был удален элемент, ссылку на добавленный или удаленный элемент;
 - CollectionReferenceChanged, которое происходит, когда одной из ссылок, входящих в коллекцию, присваивается новое значение; через объект

- CollectionHandlerEventArgs событие передает имя коллекции, строку с информацией о том, что был заменен элемент в коллекции, и ссылку на новый элемент.
9. Событие CollectionCountChanged бросают следующие методы класса MyNewCollection
 - AddDefaults();
 - Add (object[]) ;
 - Remove (int index).
 10. Событие CollectionReferenceChanged бросает метод set индексатора, определенного в классе MyNewCollection.
 11. Информация об изменениях коллекции записывается в класс Journal, который хранит информацию в списке объектов типа JournalEntry. Каждый объект типа JournalEntry содержит информацию об отдельном изменении, которое произошло в коллекции. JournalEntry содержит:
 - открытое автореализуемое свойство типа string с названием коллекции, в которой произошло событие;
 - открытое автореализуемое свойство типа string с информацией о типе изменений в коллекции;
 - открытое автореализуемое свойство типа string с данными объекта, с которым связаны изменения в коллекции;
 - конструктор для инициализации полей класса;
 - перегруженную версию метода string ToString().
 - всех элементах массива.
 12. Написать демонстрационную программу, в которой:
 - создать две коллекции MyNewCollection.
 - Создать два объекта типа Journal, один объект Journal подписать на события CollectionCountChanged и CollectionReferenceChanged из первой коллекции, другой объект Journal подписать на события CollectionReferenceChanged из обеих коллекций.
 13. Внести изменения в коллекции MyNewCollection
 - добавить элементы в коллекции;
 - удалить некоторые элементы из коллекций;
 - присвоить некоторым элементам коллекций новые значения.
 14. Вывести данные обоих объектов Journal.

Содержание отчета:

1. Диаграмма классов.
2. Определение делегата и событий.
3. Определение функций, генерирующих события.
4. Определение функций подписанных на событие.
5. Операторы, которые выполняют подписку функции на событие.
6. Текст демонстрационной программы.

Результаты работы

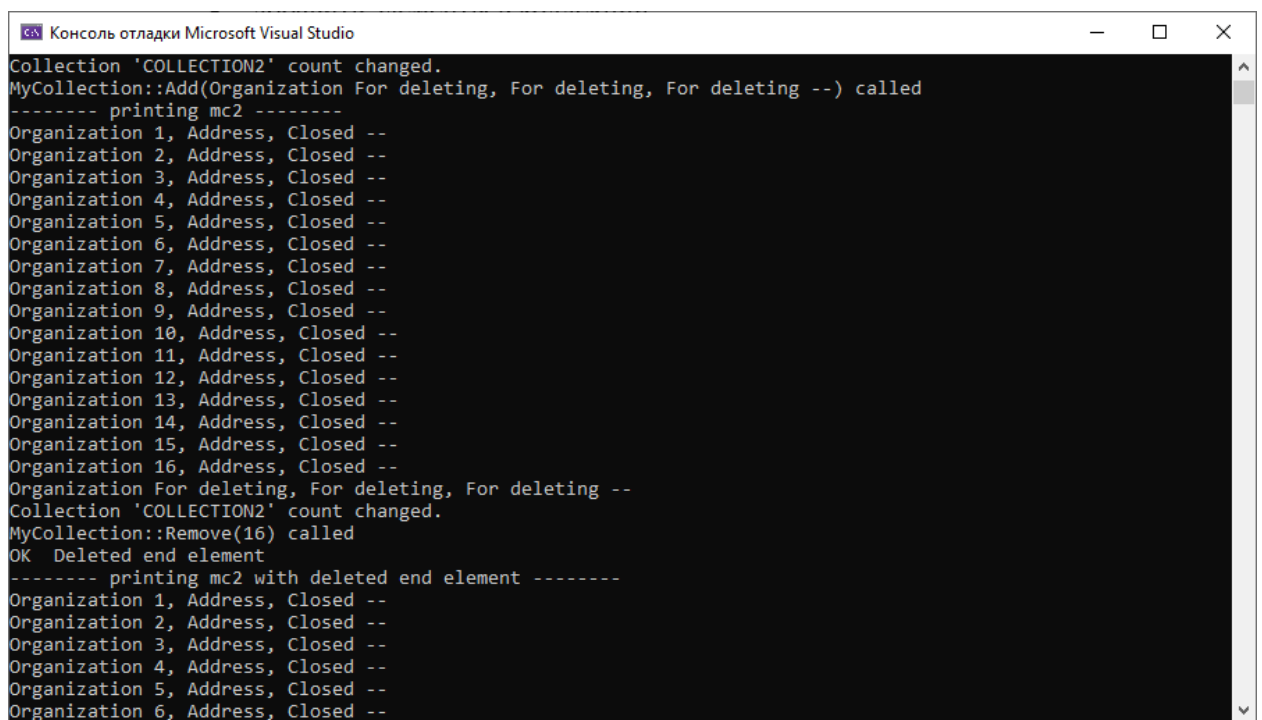


```
Консоль отладки Microsoft Visual Studio

Collection 'FIRST' count changed.
MyCollection::Add(Added org, address, time --) called
Collection 'FIRST' count changed.
MyCollection::Add(Org for delete, address, 10:00-20:00 --) called
[0]: Added org, address, time --
[1]: Org for delete, address, 10:00-20:00 --
[0]: Added org, address, time --
[1]: Org for delete, address, 10:00-20:00 --
Reference changed in collection 'FIRST'.
----- printing mc1 journal -----
----- Printing journal -----
Name collection: FIRST Event name: add
Name collection: FIRST Event name: add
Name collection: FIRST Event name: changed

Collection 'COLLECTION2' count changed.
MyCollection::Add(Organization 1, Address, Closed --) called
Collection 'COLLECTION2' count changed.
MyCollection::Add(Organization 2, Address, Closed --) called
Collection 'COLLECTION2' count changed.
MyCollection::Add(Organization 3, Address, Closed --) called
Collection 'COLLECTION2' count changed.
MyCollection::Add(Organization 4, Address, Closed --) called
Collection 'COLLECTION2' count changed.
MyCollection::Add(Organization 5, Address, Closed --) called
Collection 'COLLECTION2' count changed.
```

Рисунок 1 - Добавление элементов для фиксации изменений в журнале



```
Консоль отладки Microsoft Visual Studio

Collection 'COLLECTION2' count changed.
MyCollection::Add(Organization For deleting, For deleting, For deleting --) called
----- printing mc2 -----
Organization 1, Address, Closed --
Organization 2, Address, Closed --
Organization 3, Address, Closed --
Organization 4, Address, Closed --
Organization 5, Address, Closed --
Organization 6, Address, Closed --
Organization 7, Address, Closed --
Organization 8, Address, Closed --
Organization 9, Address, Closed --
Organization 10, Address, Closed --
Organization 11, Address, Closed --
Organization 12, Address, Closed --
Organization 13, Address, Closed --
Organization 14, Address, Closed --
Organization 15, Address, Closed --
Organization 16, Address, Closed --
Organization For deleting, For deleting, For deleting --
Collection 'COLLECTION2' count changed.
MyCollection::Remove(16) called
OK Deleted end element
----- printing mc2 with deleted end element -----
Organization 1, Address, Closed --
Organization 2, Address, Closed --
Organization 3, Address, Closed --
Organization 4, Address, Closed --
Organization 5, Address, Closed --
Organization 6, Address, Closed --
```

Рисунок 2 - Добавление 17 элементов в коллекцию а далее удаление последнего элемента из коллекции

```
Консоль отладки Microsoft Visual Studio

MyCollection::Remove(16) called
OK Deleted end element
----- printing mc2 with deleted end element -----
Organization 1, Address, Closed --
Organization 2, Address, Closed --
Organization 3, Address, Closed --
Organization 4, Address, Closed --
Organization 5, Address, Closed --
Organization 6, Address, Closed --
Organization 7, Address, Closed --
Organization 8, Address, Closed --
Organization 9, Address, Closed --
Organization 10, Address, Closed --
Organization 11, Address, Closed --
Organization 12, Address, Closed --
Organization 13, Address, Closed --
Organization 14, Address, Closed --
Organization 15, Address, Closed --
Organization 16, Address, Closed --
----- printing mc2 journal -----
----- Printing journal -----
Name collection: COLLECTION2 Event name: add
Name collection: COLLECTION2 Event name: add
Name collection: COLLECTION2 Event name: add
Name collection: COLLECTION2 Event name: add
Name collection: COLLECTION2 Event name: add
Name collection: COLLECTION2 Event name: add
Name collection: COLLECTION2 Event name: add
Name collection: COLLECTION2 Event name: add
Name collection: COLLECTION2 Event name: add
Name collection: COLLECTION2 Event name: add
```

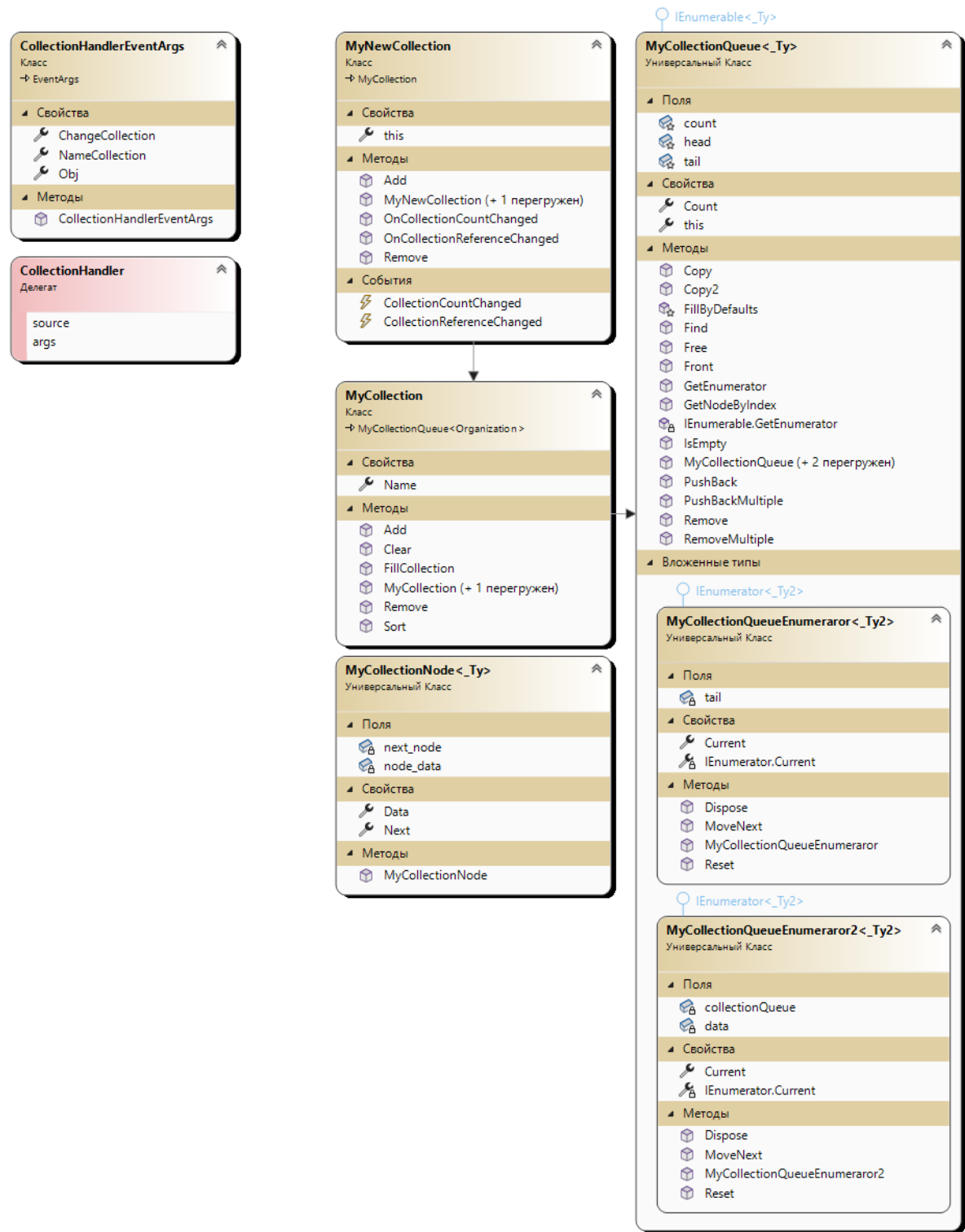
Рисунок 3 - Вывод данных после удаления последнего элемента второй коллекции

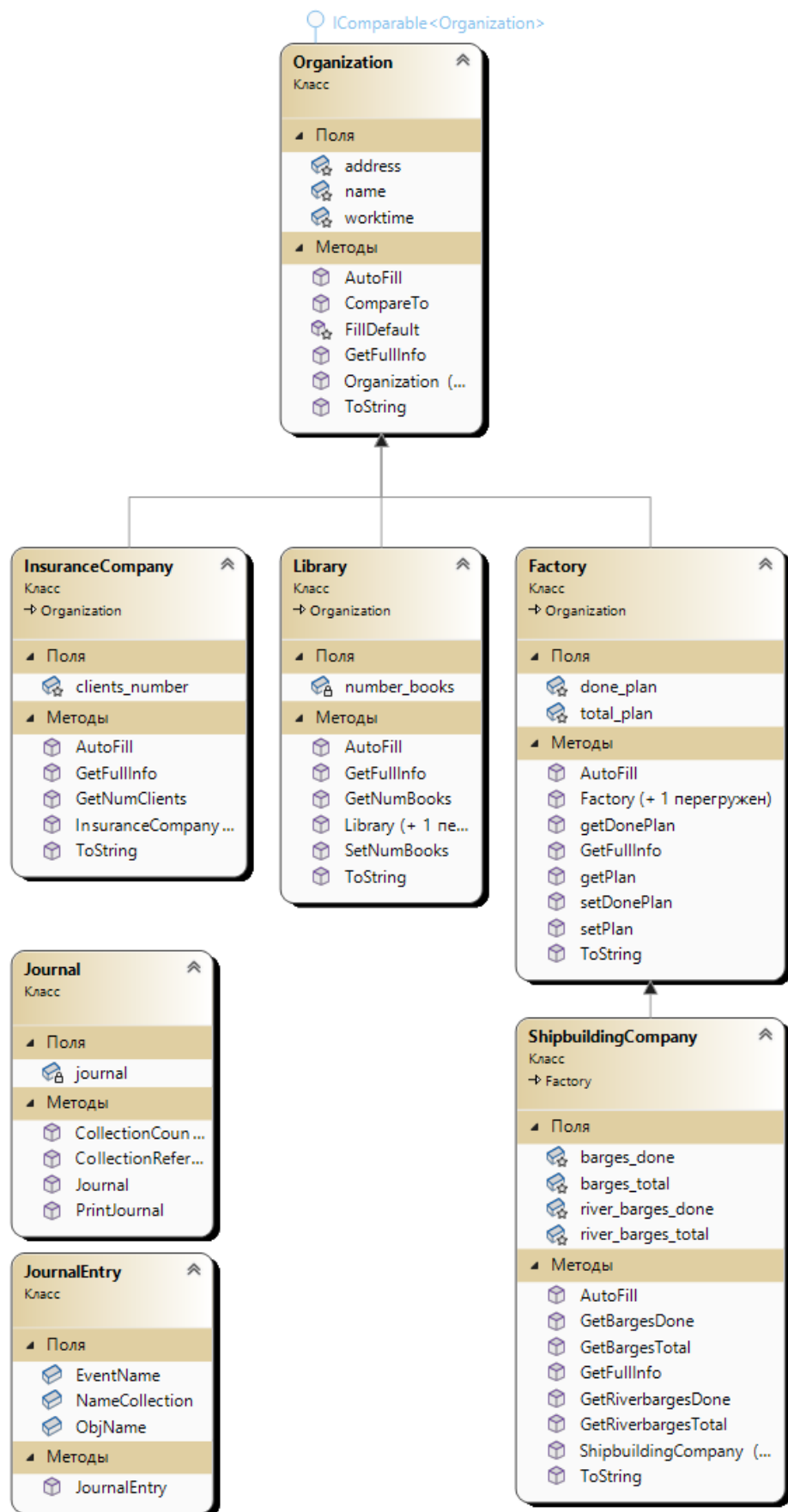
```
Консоль отладки Microsoft Visual Studio

Organization 7, Address, Closed --
Organization 8, Address, Closed --
Organization 9, Address, Closed --
Organization 10, Address, Closed --
Organization 11, Address, Closed --
Organization 12, Address, Closed --
Organization 13, Address, Closed --
Organization 14, Address, Closed --
Organization 15, Address, Closed --
Organization 16, Address, Closed --
----- printing mc2 journal -----
----- Printing journal -----
Name collection: COLLECTION2 Event name: add
Name collection: COLLECTION2 Event name: add
Name collection: COLLECTION2 Event name: add
Name collection: COLLECTION2 Event name: add
Name collection: COLLECTION2 Event name: add
Name collection: COLLECTION2 Event name: add
Name collection: COLLECTION2 Event name: add
Name collection: COLLECTION2 Event name: add
Name collection: COLLECTION2 Event name: add
Name collection: COLLECTION2 Event name: add
Name collection: COLLECTION2 Event name: add
Name collection: COLLECTION2 Event name: add
Name collection: COLLECTION2 Event name: add
Name collection: COLLECTION2 Event name: add
Name collection: COLLECTION2 Event name: add
Name collection: COLLECTION2 Event name: add
Name collection: COLLECTION2 Event name: add
Name collection: COLLECTION2 Event name: add
Name collection: COLLECTION2 Event name: delete
```

Рисунок 4 - Вывод журнала действий со второй коллекцией

Диаграмма классов





Исходный код

CollectionHandlerEventArgs.cs

```
namespace lab13k2
{
    public class CollectionHandlerEventArgs : System.EventArgs
    {
        public string NameCollection { get; set; }
        public string ChangeCollection { get; set; }
        public object Obj { get; set; }

        public CollectionHandlerEventArgs(string collectionName, string eventName,
object obj)
        {
            this.NameCollection = collectionName;
            this.ChangeCollection = eventName;
            this.Obj = obj;
        }
    }
}
```

Factory.cs

```
namespace lab13k2
{
    public class Factory : Organization
    {
        protected int done_plan;
        protected int total_plan;

        public Factory() {
            total_plan = 0;
            done_plan = 0;
        }

        public Factory(string name, string address, string worktime, int plan, int
done_plan) : base(name, address, worktime)
        {
            this.total_plan = plan;
            this.done_plan = done_plan;
        }

        public int getPlan() { return total_plan; }
        public void setPlan(int plan) { this.total_plan = plan; }
        public int getDonePlan() { return done_plan; }
        public void setDonePlan(int plan) { done_plan = plan; }

        public new string GetFullInfo()
        {
            return $"Factory: {name}, {address}, {worktime}. (Done|Total)
All({done_plan}|{total_plan})";
        }

        public new void AutoFill()
        {
            FillDefault("Factory");
        }

        public override string ToString()
        {
            return GetFullInfo();
        }
    }
}
```

InsuranceCompany.cs

```
namespace lab13k2
{
    public class InsuranceCompany : Organization
    {
        protected int clients_number;

        public InsuranceCompany() {
            clients_number = 0;
        }

        public InsuranceCompany(string name, string address, string worktime, int
clients) : base(name, address, worktime)
        {
            clients_number = clients;
        }

        public int GetNumClients() { return clients_number; }
        public new string GetFullInfo()
        {
            return $"InsuranceCompany: {name}, {address}, {worktime}.
Clients({clients_number})";
        }

        public new void AutoFill()
        {
            FillDefault("InsuranceCompany");
        }

        public override string ToString()
        {
            return GetFullInfo();
        }
    }
}
```

Journal.cs

```
namespace lab13k2
{
    public class Journal
    {
        List<JournalEntry> journal;

        public Journal() {
            journal = new List<JournalEntry>();
        }

        public void CollectionCountChanged(object sender,
CollectionHandlerEventArgs e) {
            Console.WriteLine($"Collection '{e.NameCollection}' count changed.");
            JournalEntry je = new JournalEntry(e.NameCollection,
e.ChangeCollection, e.Obj.ToString());
            journal.Add(je);
        }

        public void CollectionReferenceChanged(object sender,
CollectionHandlerEventArgs e) {
            Console.WriteLine($"Reference changed in collection
'{e.NameCollection}'.");
            JournalEntry je = new JournalEntry(e.NameCollection,
e.ChangeCollection, e.Obj.ToString());
            journal.Add(je);
        }

        public void PrintJournal() {
            Console.WriteLine("----- Printing journal -----");
            foreach(JournalEntry je in journal) {
```



```

        Console.WriteLine("Name collection: {0} Event name: {1}",
je.NameCollection, je.EventName);
    }
}
}

```

JournalEntry.cs

```

namespace lab13k2
{
    public class JournalEntry
    {
        public string NameCollection;
        public string EventName;
        public string ObjName;

        public JournalEntry(string name, string eventname, string objName) {
            NameCollection = name;
            EventName = eventname;
            ObjName = objName;
        }
    }
}

```

Lab.cs

```

namespace lab13k2
{
    public class Lab
    {
        MyNewCollection mc1;
        MyNewCollection mc2;

        public void Start() {
            mc1 = new MyNewCollection("FIRST");
            Journal joun1 = new Journal();
            mc1.CollectionCountChanged += new
CollectionHandler(joun1.CollectionCountChanged);
            mc1.CollectionReferenceChanged += new
CollectionHandler(joun1.CollectionReferenceChanged);

            Organization orgForDel = new Organization("Org for delete", "address",
"10:00-20:00");
            mc1.Add(new Organization("Added org", "address", "time"));
            mc1.Add(orgForDel);

            int j = 0;
            foreach (var i in mc1)
                Console.WriteLine("[{0}]: {1}", j++, i.GetFullInfo());

            MyCollectionNode<Organization> ?node = mc1.Find(orgForDel);
            if(mc1.Remove(node)) {
                Console.WriteLine("element removed from collection\n");
            }

            j = 0;
            foreach (var i in mc1)
                Console.WriteLine("[{0}]: {1}", j++, i.GetFullInfo());

            mc1[0] = orgForDel;

            Console.WriteLine("----- printing mc1 journal -----");
            joun1.PrintJournal();

            Console.WriteLine("\n\n\n\n");

            /***** collection 2 *****/

```

```

        mc2 = new MyNewCollection("COLLECTION2");
        Journal joun2 = new Journal();
        mc2.CollectionCountChanged += new
CollectionHandler(joun2.CollectionCountChanged);
        mc2.CollectionReferenceChanged += new
CollectionHandler(joun2.CollectionReferenceChanged);
        for (int i = 0; i < 16; i++) {
            Organization org = new Organization($"Organization {i + 1}",
"Address", "Closed");
            mc2.Add(org);
        }

        Organization orgForDeletingFromMc2 = new Organization("Organization
For deleting", "For deleting", "For deleting");
        mc2.Add(orgForDeletingFromMc2);

        Console.WriteLine("----- printing mc2 -----");
        foreach (var i in mc2)
            Console.WriteLine(i.GetFullInfo());

        if (mc2.Remove(mc2.Count - 1))
            Console.WriteLine("OK Deleted end element");

        Console.WriteLine("----- printing mc2 with deleted end element ----
----");
        foreach (var i in mc2)
            Console.WriteLine(i.GetFullInfo());

        Console.WriteLine("----- printing mc2 journal -----");
        joun2.PrintJournal();
    }
}

```

Library.cs

```
namespace lab13k2
```

```

{
    public class Library : Organization
    {
        private int number_books;

        public Library() {
            number_books = 0;
        }

        public Library(string name, string address, string worktime, int
num_books) : base(name, address, worktime)
        {
            number_books = num_books;
        }

        public int GetNumBooks() { return number_books; }
        public void SetNumBooks(int n) { number_books = n; }

        public new string GetFullInfo()
        {
            return $"Library: {name}, {address}, {worktime}. Number of books:
{number_books}";
        }

        public new void AutoFill()
        {
            FillDefault("Library");
        }

        public override string ToString()
        {

```

```

        return GetFullInfo();
    }
}

```

MyCollection.cs

```

namespace lab13k2
{
    public delegate void CollectionHandler(object source,
    CollectionHandlerEventArgs args); //делегат

    public class MyCollection : MyCollectionQueue<Organization>
    {
        public string Name { get; set; }

        public MyCollection(string name) : base() {
            Name = name;
        }

        public MyCollection(string name, int cap) : base(cap) {
            Name = name;
        }

        public virtual bool Remove(int postition)
        {
            Console.WriteLine("MyCollection::Remove({0}) called", postition);
            MyCollectionNode<Organization>? node = GetNodeByIndex(postition);
            return base.Remove(node);
        }

        public virtual bool Add(Organization org)
        {
            Console.WriteLine("MyCollection::Add({0}) called", org);
            base.PushBack(org);
            return true;
        }

        public void FillCollection() {
            MyCollectionNode<Organization>? node = tail;
            while (node != null) {
                node.Data.AutoFill();
                node = node.Next;
            }
        }

        public void Sort() {
            MyCollectionNode<Organization>? node = tail;
            List<Organization> list = new List<Organization>(count);
            while (node != null) {
                list.Add(node.Data);
                node = node.Next;
            }

            list.Sort();

            node = tail;
            foreach (var item in list) {
                Debug.Assert(node != null, "node can never be null here!");
                node.Data = item;
                node = node.Next;
            }
        }

        public void Clear() {
            Free();
        }
    }
}

```

```

    }
}

```

MyCollectionQueue.cs

```

namespace lab13k2
{
    public class MyCollectionNode<_Ty>
    {
        MyCollectionNode<_Ty>? next_node;
        _Ty node_data;

        public MyCollectionNode(MyCollectionNode<_Ty>? next, _Ty data) {
            next_node = next;
            node_data = data;
        }

        public _Ty Data {
            get { return node_data; }
            set { node_data = value; }
        }

        public MyCollectionNode<_Ty>? Next {
            get { return next_node; }
            set { next_node = value; }
        }
    };

    public class MyCollectionQueue<_Ty> : IEnumerable<_Ty>
    {
        public class MyCollectionQueueEnumeraror<_Ty2> : IEnumerator<_Ty2>
        {
            private MyCollectionNode<_Ty2>? tail;

            public MyCollectionQueueEnumeraror(MyCollectionNode<_Ty2>? tailref) {
                tail = tailref;
            }

            public _Ty2 Current {
                get { return tail.Data; }
            }
            object IEnumerator.Current {
                get { return Current; }
            }
            public bool MoveNext()
            {
                if (tail != null)
                    tail = tail.Next;

                return tail != null;
            }
            public void Reset()
            {
                tail = null;
            }
            public void Dispose() { }
        };

        public class MyCollectionQueueEnumeraror2<_Ty2> : IEnumerator<_Ty2>
        {
            _Ty2 data;
            MyCollectionQueue<_Ty2> collectionQueue;

            public MyCollectionQueueEnumeraror2(MyCollectionQueue<_Ty2>?
thisQueue) {

```

```

        collectionQueue = thisQueue.Copy2();
    }

    public _Ty2 Current
    {
        get { return data; }
    }
    object IEnumerator.Current
    {
        get { return Current; }
    }
    public bool MoveNext()
    {
        if (!collectionQueue.IsEmpty()) {
            data = collectionQueue.Front();
            return true;
        }
        return false;
    }
    public void Reset() { }
    public void Dispose() { }
};

public IEnumerator<_Ty> GetEnumerator() {
    //return new MyCollectionQueueEnumeraror<_Ty>(tail);
    return new MyCollectionQueueEnumeraror2<_Ty>(this);
}

IEnumerator IEnumerable.GetEnumerator() {
    return GetEnumerator();
}

protected int count;
protected MyCollectionNode<_Ty>? head;
protected MyCollectionNode<_Ty>? tail;

public int Count {
    get { return count; }
}

public MyCollectionQueue() {
    head = null;
    tail = null;
    count = 0;
}

protected void FillByDefaults() {
    // if number of elements greater 0
    if (count > 0)
    {
        _Ty data = default(_Ty);
        MyCollectionNode<_Ty>? newNode = new MyCollectionNode<_Ty>(null,
data);

        // create new empty nodes
        for (int i = 0; i < count; i++)
        {
            if (head != null) // if head exists element
                head.Next = newNode; // set next ref to exists element

            head = newNode;
        }
    }
}

public MyCollectionQueue(int capacity) {
    head = null;
    tail = null;
}

```

```

        count = capacity; // define queue size
        FillByDefaults();
    }

    public MyCollectionQueue(MyCollectionQueue<_Ty> ?queueWithInit) {
        if(queueWithInit != null) {
            MyCollectionQueue<_Ty> copy = queueWithInit.Copy2();
            while (!copy.IsEmpty()) {
                PushBack(copy.Front());
            }
        }
    }

    public void PushBack(_Ty data) {
        MyCollectionNode<_Ty>? newNode = null;
        newNode = new MyCollectionNode<_Ty>(null, data);
        if (head != null) // if previous node exists
            head.Next = newNode; // next node for previous - this new node

        head = newNode; //set new node to head ref
        if (tail == null)
            tail = head; // queue is empty or not initialized. Set tail to
head ref

        count++; // increment count elements in queue
    }

    public void PushBackMultiple(_Ty[] dataArray, int count) {
        if(count > 0) {
            for (int i = 0; i < count; i++) {
                PushBack(dataArray[i]);
            }
        }
    }

    public MyCollectionNode<_Ty>? Find(_Ty dataForFind) {
        MyCollectionNode<_Ty>? nodeRef = tail;
        if (nodeRef != null) {
            while(nodeRef != null) {
                if(nodeRef.GetHashCode() == dataForFind.GetHashCode()) {
                    return nodeRef; // element found
                }
                nodeRef = nodeRef.Next;
            }
        }
        return null; // not found
    }

    public bool Remove(MyCollectionNode<_Ty>? nodeRefForDel) {
        if (nodeRefForDel == null)
            return false;

        MyCollectionNode<_Ty>? nodeRef = tail; // tail is start
        while (nodeRef != null) { // if start node is not null
            MyCollectionNode<_Ty>? nextRef = nodeRef.Next; // save ref to next
node

            if(nextRef != null) { // if ref to next node is not null
                if (nodeRefForDel == nextRef) { // if ref to next node equals
ref node for delete
                    nodeRef.Next = nodeRefForDel.Next; // set 'next' this node
ref to 'next' node ref in deleting
                    return true;
                }
            }
            nodeRef = nextRef;
        }
        return false;
    }

```

```

    }

    public bool RemoveMultiple(MyCollectionNode<_Ty>?[] nodesRefForDel, int
count) {
        bool bSuccess = true; // return is OK
        for (int i = 0; i < count; i++) // for each element
            bSuccess &= Remove(nodesRefForDel[i]); // change bSuccess to false
        if one of function failed

        return bSuccess; // return bSuccess
    }

    public bool IsEmpty() {
        return tail == null; // tail ref is null. queue is empty
    }

    public _Ty Front() {
        _Ty data = default(_Ty); // init new empty object instance data
        if (tail != null) { // if tail not null
            data = tail.Data; // copy data from queue node
            tail = tail.Next; // move to next ref and set tail to this ref
            count--; // element readed from queue, decrement count
        }
        return data; // return copied data
    }

    public MyCollectionQueue<_Ty> Copy() { // DEPTH copy
        return new MyCollectionQueue<_Ty>(this);
    }

    public MyCollectionQueue<_Ty> Copy2() {
        MyCollectionQueue<_Ty> queueCopy = new MyCollectionQueue<_Ty>();
        queueCopy.head = head;
        queueCopy.tail = tail;
        queueCopy.count = count;
        return queueCopy;
    }

    // free memory
    public void Free() {
        head = null;
        tail = null;
    }

    public MyCollectionNode<_Ty>? GetNodeByIndex(int index)
    {
        int i = 0;
        MyCollectionNode<_Ty>? nodeRef = tail;
        while (nodeRef != null)
        {
            if (index == i)
                return nodeRef;

            i++;
            nodeRef = nodeRef.Next;
        }
        return null;
    }

    public virtual _Ty this[int index] {
        get {
            MyCollectionNode<_Ty>? nodeRef = GetNodeByIndex(index);
            if (nodeRef == null)
                throw new IndexOutOfRangeException($"index {index} out of
bounds");

            return nodeRef.Data;
        }
    }

```

```

    }
    set
    {
        MyCollectionNode<_Ty>? nodeRef = GetNodeByIndex(index);
        if (nodeRef == null)
            throw new IndexOutOfRangeException($"index {index} out of
bounds");

        nodeRef.Data = value;
    }
}
}
}

```

MyNewCollection.cs

```

namespace lab13k2
{
    public class MyNewCollection : MyCollection
    {
        public MyNewCollection(string name) : base(name) {
        }

        public MyNewCollection(string name, int cap) : base(name, cap) {
        }

        public event CollectionHandler CollectionCountChanged;
        public event CollectionHandler CollectionReferenceChanged;

        //обработчик события CollectionCountChanged
        public virtual void OnCollectionCountChanged(object source,
CollectionHandlerEventArgs args)
        {
            if (CollectionCountChanged != null)
                CollectionCountChanged(source, args);
        }
        //обработчик события OnCollectionReferenceChanged
        public virtual void OnCollectionReferenceChanged(object source,
CollectionHandlerEventArgs args)
        {
            if (CollectionReferenceChanged != null)
                CollectionReferenceChanged(source, args);
        }

        public override bool Remove(int position)
        {
            OnCollectionCountChanged(this, new
CollectionHandlerEventArgs(this.Name, "delete", this[position]));
            return base.Remove(position);
        }

        public override bool Add(Organization p)
        {
            OnCollectionCountChanged(this, new
CollectionHandlerEventArgs(this.Name, "add", p));
            return base.Add(p);
        }

        public override Organization this[int index]
        {
            get
            {
                return base[index];
            }
            set
            {

```



```

        OnCollectionReferenceChanged(this, new
CollectionHandlerEventArgs(this.Name, "changed", this[index]));
        base[index] = value;
    }
}
}
}

```

Organization.cs

```

namespace lab13k2
{
    public class Organization : IComparable<Organization>
    {
        protected string name;
        protected string address;
        protected string worktime;

        protected void FillDefault(string prefix)
        {
            Random rand = new Random();
            name = $"{prefix}_{rand.Next()}";
            address = $"address {rand.Next(0, 2000)}";
            int startHour = rand.Next(8, 12);
            int startMins = rand.Next(0, 30);
            int endHour = rand.Next(startHour + 1, startHour + 8) % 24;
            int endMins = rand.Next(0, 30);
            worktime = $"{startHour}:{startMins}-{endHour}:{endMins}";
        }

        public Organization()
        {
            name = "empty";
            address = "empty";
            worktime = "empty";
        }

        public int CompareTo(Organization? other)
        {
            return name.CompareTo(other.name);
        }

        public Organization(string _name, string _address, string _worktime)
        {
            name = _name;
            address = _address;
            worktime = _worktime;
        }

        public string GetFullInfo() { return $"{name}, {address}, {worktime} --"; }

        public void AutoFill()
        {
            FillDefault("Organization");
        }

        public override string ToString()
        {
            return GetFullInfo();
        }
    }
}

```

ShipbuildingCompany.cs

```
namespace lab13k2
{
    /* судостроительная */
    public class ShipbuildingCompany : Factory
    {
        protected int barges_total;
        protected int barges_done;
        protected int river_barges_total;
        protected int river_barges_done;

        public ShipbuildingCompany() {
            barges_total = 0;
            barges_done = 0;
            river_barges_total = 0;
            river_barges_done = 0;
        }

        public ShipbuildingCompany(string name, string address, string worktime,
            int barges_total, int barges_done, int rbarges_total, int rbarges_done)
            : base(name, address, worktime, barges_total + rbarges_total,
            barges_done + rbarges_done)
        {
            this.barges_total = barges_total;
            this.barges_done = barges_done;
            this.river_barges_total = rbarges_total;
            this.river_barges_done = rbarges_done;
        }

        public int GetBargesTotal() { return barges_total; }
        public int GetBargesDone() { return barges_done; }
        public int GetRiverbargesTotal() { return river_barges_total; }
        public int GetRiverbargesDone() { return river_barges_done; }

        public new string GetFullInfo()
        {
            return $"ShipbuildingCompany: {name}, {address}, {worktime}
            (Total|Done) All({total_plan}|{done_plan}) barges({barges_total}|{barges_done})
            riverbarges({river_barges_total}|{river_barges_done})";
        }

        public new void AutoFill()
        {
            FillDefault("ShipbuildingCompany");
        }

        public override string ToString()
        {
            return GetFullInfo();
        }
    }
}
```

Program.cs

```
namespace lab13k2
{
    internal class Program
    {
        static void Main(string[] args) {
            new Lab().Start();
        }
    }
}
```