**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 Сollection<MyClass>.
3. Класс Сollection<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.

1. 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.

1. Для событий, извещающих об изменениях в коллекции, определяется свой делегат CollectionHandler с сигнатурой:

void CollectionHandler (object source, CollectionHandlerEventArgs args);

1. 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.

1. 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 cобытие 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.

1. Событие CollectionCountChanged бросают следующие методы класса MyNewCollection

* AddDefaults();
* Add (object[] ) ;
* Remove (int index).

1. Событие CollectionReferenceChanged бросает метод set индексатора, определенного в классе MyNewCollection.
2. 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 c данными объекта, с которым связаны изменения в коллекции;
* конструктор для инициализации полей класса;
* перегруженную версию метода string ToString().
* всех элементах массива.

1. 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.

1. Внести изменения в коллекции MyNewCollection

* добавить элементы в коллекции;
* удалить некоторые элементы из коллекций;
* присвоить некоторым элементам коллекций новые значения.

1. 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**

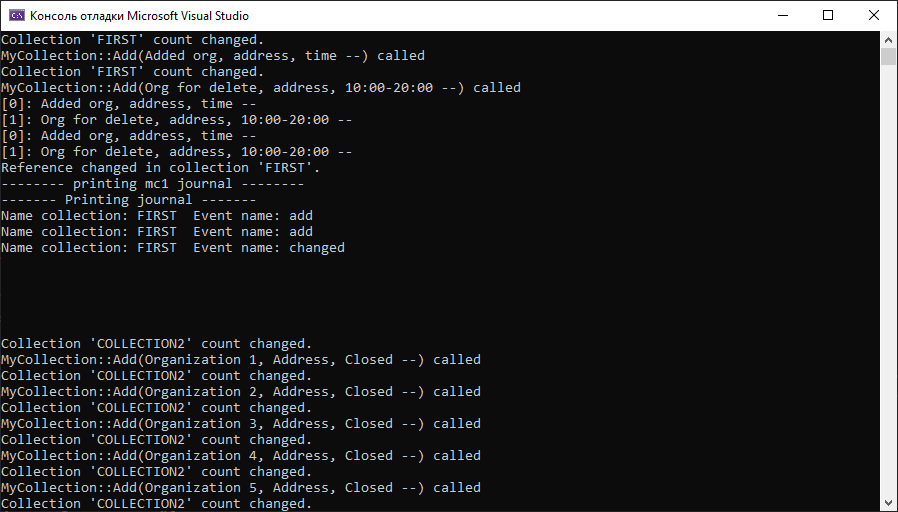


Рисунок - Добавление элементов для фиксации изменений в журнале

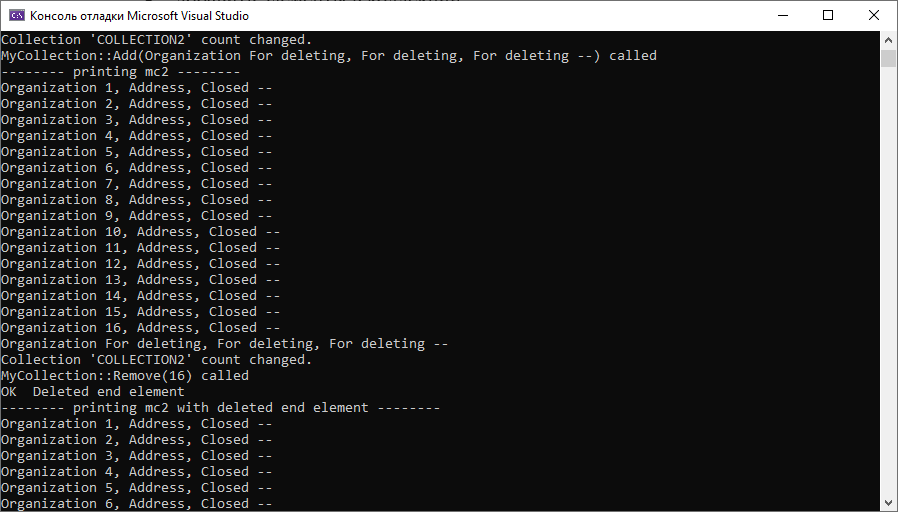


Рисунок - Добавление 17 элементов в коллекцию а далее удаление последнего элемента из коллекции

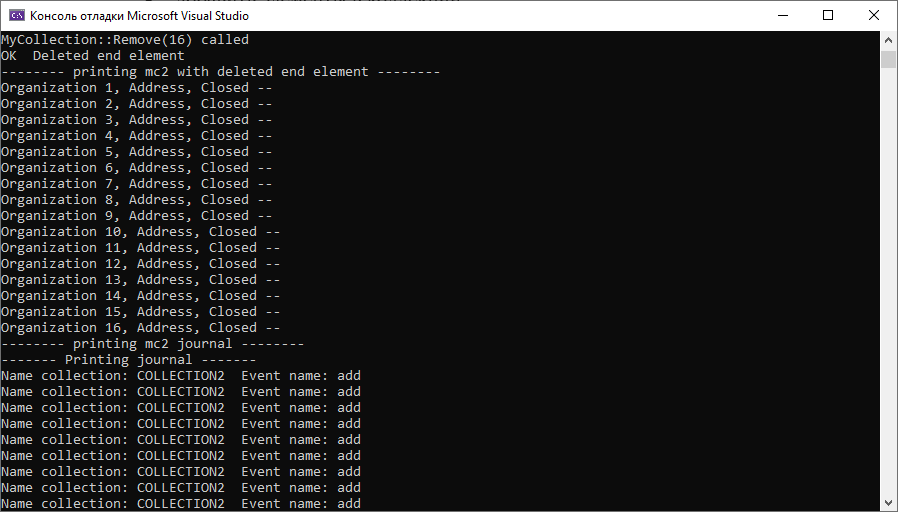


Рисунок - Вывод данных после удаления последнего элемента второй коллекции

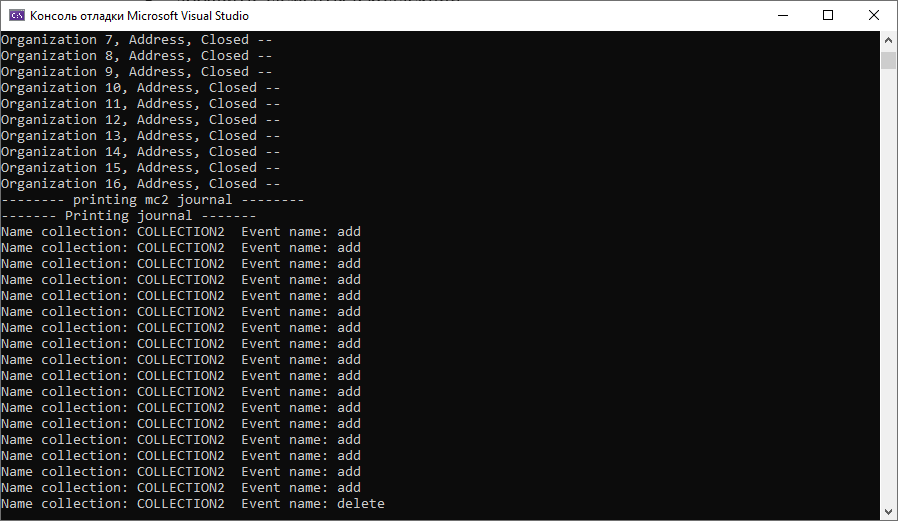
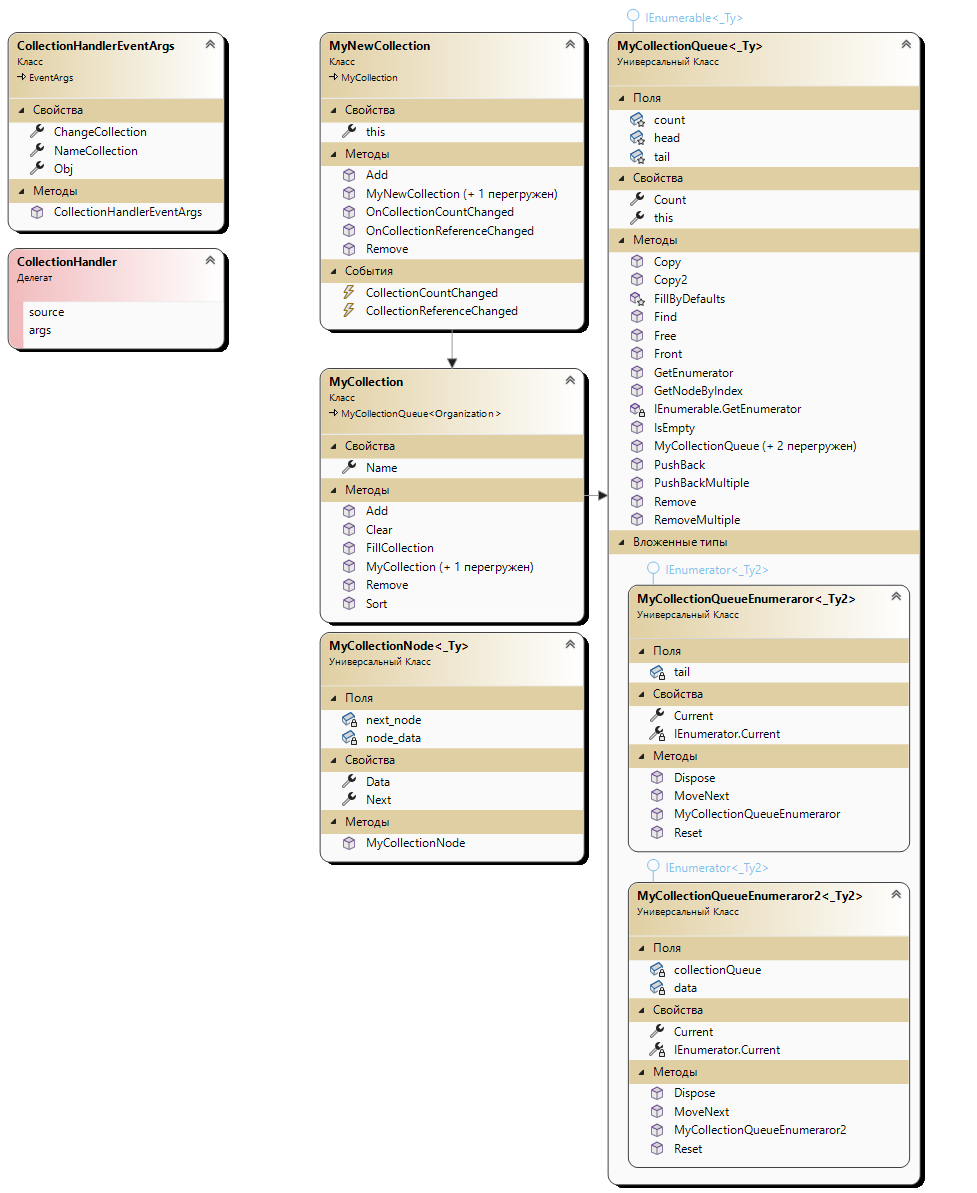
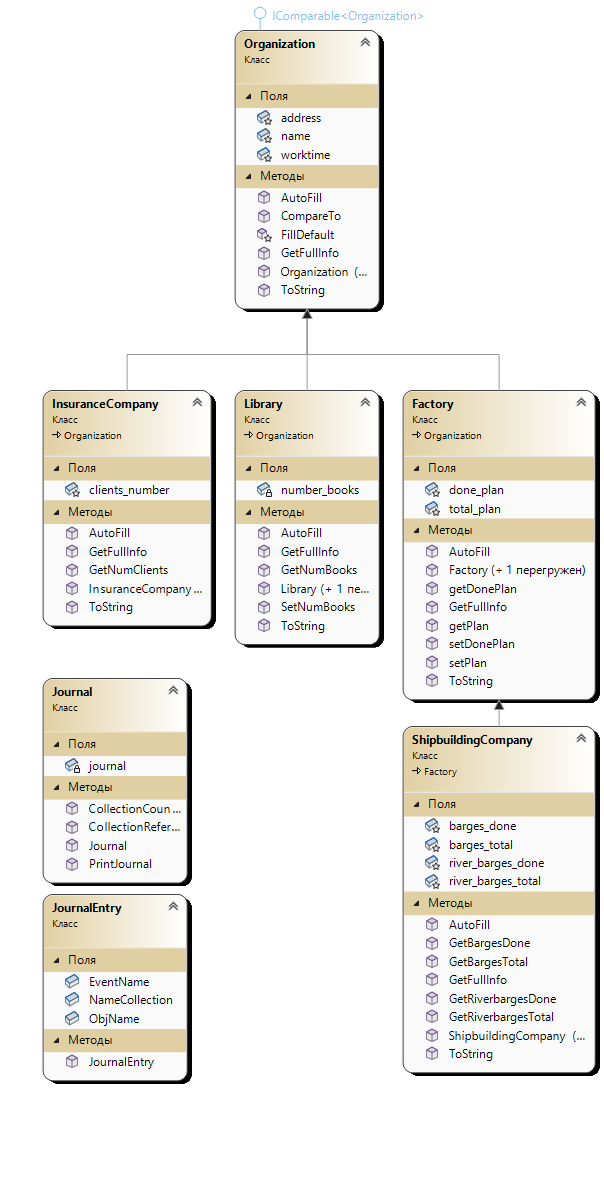


Рисунок - Вывод журнала действий со второй коллекцией

**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();

}

}

}