|  |
| --- |
| Пермский филиал федерального государственного автономного образовательного учреждения высшего образования  «Национальный исследовательский университет  «Высшая школа экономики»  *Факультет социально-экономических и компьютерных наук* |
|  |
| Гуцол Степан Дмитриевич  **ЛАБОРАТОРНАЯ РАБОТА №13**  по направлению подготовки *38.03.05 Бизнес-информатика*  образовательная программа «Разработка информационных систем для бизнеса»   |  |  | | --- | --- | |  | Руководитель  Преподаватель кафедры ИТБ  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Найданов И.В. |   Пермь, 2023 |

**Оглавление**

[Часть №1 Двунаправленный список 4](#_Toc136615889)

[1 Постановка задачи 4](#_Toc136615890)

[2 Диаграмма классов 5](#_Toc136615891)

[3 Программа (листинг) 6](#_Toc136615892)

[4 Код Unit-тестов 7](#_Toc136615893)

[5 Анализ покрытия кода тестами 8](#_Toc136615894)

[Часть №2 Идеально-сбалансированное дерево и дерево поиска 10](#_Toc136615895)

[1 Постановка задачи 10](#_Toc136615896)

[2 Диаграмма классов 11](#_Toc136615897)

[3 Программа (листинг) 12](#_Toc136615898)

[4 Код Unit-тестов 13](#_Toc136615899)

[5 Анализ покрытия кода тестами 14](#_Toc136615900)

[Часть №3 Хэш-таблица с открытой адресацией 16](#_Toc136615901)

[1 Постановка задачи 16](#_Toc136615902)

[2 Диаграмма классов 17](#_Toc136615903)

[3 Программа (листинг) 18](#_Toc136615904)

[4 Код Unit-тестов 19](#_Toc136615905)

[5 Анализ покрытия кода тестами 20](#_Toc136615906)

[Часть №4 Реализация коллекции (дерево поиска) 22](#_Toc136615907)

[1 Постановка задачи 22](#_Toc136615908)

[2 Диаграмма классов 23](#_Toc136615909)

[3 Программа (листинг) 24](#_Toc136615910)

[4 Код Unit-тестов 25](#_Toc136615911)

[5 Анализ покрытия кода тестами 26](#_Toc136615912)

[ПРИЛОЖЕНИЕ А Программа Program.cs (часть 1) 27](#_Toc136615913)

[ПРИЛОЖЕНИЕ Б Программа Point.cs (часть 1) 29](#_Toc136615914)

[ПРИЛОЖЕНИЕ В Программа Color.cs (часть 1) 32](#_Toc136615915)

[ПРИЛОЖЕНИЕ Г Программа AdditionalActions.cs (часть 1) 33](#_Toc136615916)

[ПРИЛОЖЕНИЕ Д Программа Program.cs (часть 2) 34](#_Toc136615917)

[ПРИЛОЖЕНИЕ Е Программа Point.cs (часть 2) 35](#_Toc136615918)

[ПРИЛОЖЕНИЕ Ё Программа MenuActivity.cs (часть 2) 37](#_Toc136615919)

[ПРИЛОЖЕНИЕ Ж Программа Tree.cs (часть 2) 39](#_Toc136615920)

[ПРИЛОЖЕНИЕ З Программа AdditionalActions.cs (часть 2) 44](#_Toc136615921)

[ПРИЛОЖЕНИЕ И Программа Program.cs (часть 3) 46](#_Toc136615922)

[ПРИЛОЖЕНИЕ Й Программа Point.cs (часть 3) 47](#_Toc136615923)

[ПРИЛОЖЕНИЕ К Программа MenuActivity.cs (часть 3) 48](#_Toc136615924)

[ПРИЛОЖЕНИЕ Л Программа HTable.cs (часть 3) 49](#_Toc136615925)

[ПРИЛОЖЕНИЕ М Программа AdditionalActions.cs (часть 3) 53](#_Toc136615926)

[ПРИЛОЖЕНИЕ Н Программа Program.cs (часть 4) 55](#_Toc136615927)

[ПРИЛОЖЕНИЕ О Программа Point.cs (часть 4) 57](#_Toc136615928)

[ПРИЛОЖЕНИЕ П Программа MyCollecyion.cs (часть 4) 59](#_Toc136615929)

[ПРИЛОЖЕНИЕ Р Программа MenuActivity.cs (часть 4) 68](#_Toc136615930)

[ПРИЛОЖЕНИЕ С Программа AdditionalActions.cs (часть 4) 71](#_Toc136615931)

[ПРИЛОЖЕНИЕ Т Программа UnitTest1.cs (Юнит-Тесты) 73](#_Toc136615932)

[ПРИЛОЖЕНИЕ У Покрытие кода Fine Code Coverage 75](#_Toc136615933)

**Разработка программы, управляемой событиями**

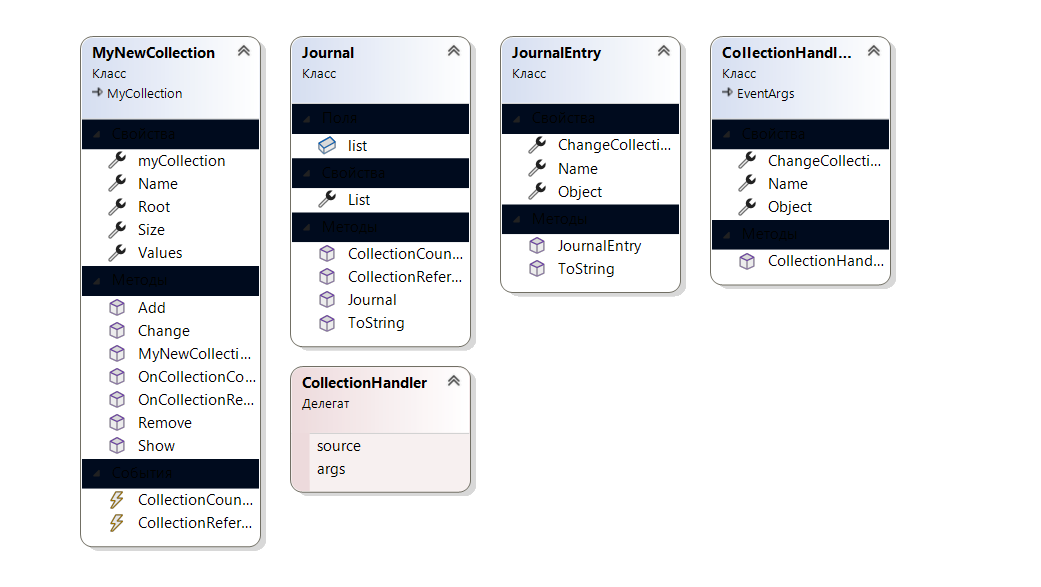
**1 Постановка задачи**

1. Создать иерархию классов. 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 (библиотека из лаб. 10).
2. Определить класс MyNewCollection<T> производный от класса MyCollection<T>. Класс MyСollection<T> взять из лабораторной работы №12 (лучше, как библиотеку). В классе MyСollection<T> должны быть реализованы методы:
   * для заполнения коллекции (элементы коллекции формируются автоматически),
   * добавления элементов коллекции,
   * удаления элементов коллекции, очистки коллекции,
   * реализован итератор для доступа к элементам коллекции,
   * индексатор (с методами get и set) с целочисленным индексом для изменения элементов коллекции,
   * свойство Length (только для чтения), содержащее текущее количество элементов коллекции.
3. В класс MyNewCollection добавить события, которые извещают 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 генерируются события.

1. Для событий, извещающих об изменениях в коллекции, определяется делегат CollectionHandler: void CollectionHandler (object source, CollectionHandlerEventArgs args);
2. Для передачи информации о событии определить класс CollectionHandlerEventArgs, производный от класса System.EventArgs, который содержит
   * 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;
3. 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 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 строку с информацией о том, что был заменен элемент в коллекции, и ссылку на новый элемент.
4. Событие CollectionCountChanged бросают следующие методы класса MyNewCollection
   * Add (Т obj ) ; //добавить объект
   * Remove (int index) //удалить элемент по индексу.
5. Событие CollectionReferenceChanged бросает метод set индексатора, определенного в классе MyNewCollection.
6. 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().
7. 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.
8. Внести изменения в коллекции MyNewCollection
   * добавить элементы в коллекции;
   * удалить некоторые элементы из коллекций;
   * присвоить некоторым элементам коллекций новые значения.
9. Вывести данные обоих объектов Journal.

**2 Диаграмма классов**



**Рисунок 1 – Диаграмма классов**

Программа состоит из четырёх классов, а именно:

* Journal – в этом классе реализованы все функции для создания и добавления элементов журнала класса JournalEntry в журнал.
* JournalEntry – Этот класс реализует дополнительные функции, необходимые для создания записи в журнал
* CollectionHandlerEventArgs – Вспомогательный класс, который хранит в себе данные.
* MyNewCollection – Класс коллекции, производный от старой.

**3 Программа (листинг)**

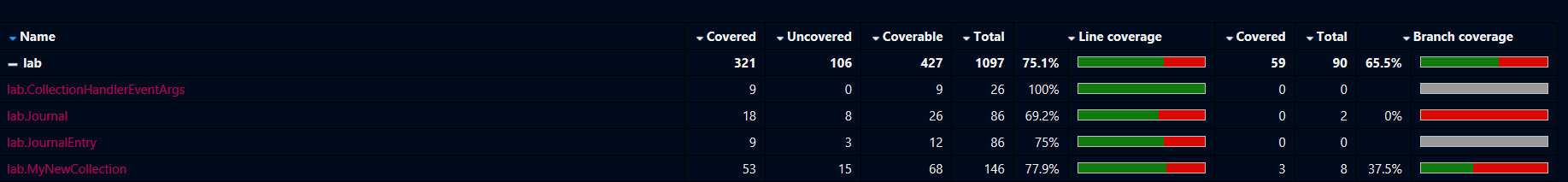
Код программы представлен в приложении. См. Приложение А-Г

**4 Код Unit-тестов**

Код программы Unit-тестов представлен в приложении. См. Приложение Д

**5 Анализ покрытия кода тестами**

С помощью Unit-тестов удалось достичь высокий процент покрытия кода, так как в этой работе большое количество методов (Рисунок 2).



**Рисунок 2 – Покрытие кода тестами**

# ПРИЛОЖЕНИЕ А **Программа Program.cs**

using lab12\_part4;

using System.Drawing;

namespace lab

{

internal class Program

{

static void Main(string[] args)

{

// Создание первой коллекции

MyNewCollection myCollection1 = new MyNewCollection("FIRST", 10);

Journal journal1 = new Journal();

myCollection1.CollectionCountChanged += new CollectionHandler(journal1.CollectionCountChanged);

myCollection1.CollectionReferenceChanged += new CollectionHandler(journal1.CollectionReferenceChanged);

// Обработка первой коллекции

myCollection1.Remove(5);

myCollection1.Add();

myCollection1.Change(7, "Test777");

myCollection1.Change(5, "Test555");

//Вывод журнала и коллекции

Console.WriteLine(journal1);

MyNewCollection.Show(myCollection1);

Console.WriteLine( );

//Создание второй коллекции

Console.ForegroundColor = ConsoleColor.Green;

MyNewCollection myCollection2 = new MyNewCollection("Second", 10);

Journal journal2 = new Journal();

myCollection2.CollectionCountChanged += new CollectionHandler(journal2.CollectionCountChanged);

myCollection2.CollectionReferenceChanged += new CollectionHandler(journal2.CollectionReferenceChanged);

//Обработка второй коллекции

myCollection2.Remove(1);

myCollection2.Remove(8);

myCollection2.Remove(9);

myCollection2.Remove(10);

//Вывод второй коллекции и второго журнала

Console.WriteLine(journal2);

MyNewCollection.Show(myCollection2);

}

}

}

# ПРИЛОЖЕНИЕ Б **Программа Journal.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab

{

/// <summary>

/// Класс объектов, храянщих данные об операциях

/// </summary>

public class JournalEntry

{

public string Name { get; set; } // Информационное поле с именем коллекции

public string ChangeCollection { get; set; } // Информционное поле с информацией об операции с коллекцией

public string Object { get; set; } // Объект, который изменяли или добавили

public JournalEntry(string Name, string ChangeCollection, string Object)

{

this.Name = Name;

this.ChangeCollection = ChangeCollection;

this.Object = Object;

}

public override string ToString()

{

return $"В коллекции {Name} произошло {ChangeCollection} объекта {Object}" +"\n";

}

}

/// <summary>

/// Класс журнала и операций с ним

/// </summary>

public class Journal

{

public List<JournalEntry> list = new List<JournalEntry>();

public List<JournalEntry> List

{

get

{

return list;

}

set

{

}

}

public Journal()

{

List = null;

}

/// <summary>

/// Перегруженная функция вывода журнала

/// </summary>

/// <returns>Возвращает строку с инфомацией об ивентах</returns>

public override string ToString()

{

string line = "";

foreach (var entry in list)

{

line += entry.ToString();

}

return line;

}

/// <summary>

/// Функция события для добавления информации в журнал о добавлении или удалении объектов

/// </summary>

/// <param name="source">Источник данных для обработки</param>

/// <param name="e">Необходимые параметры</param>

public void CollectionCountChanged(object source, CollectionHandlerEventArgs e)

{

JournalEntry je = new JournalEntry(e.Name, e.ChangeCollection, e.Object.ToString());

List.Add(je);

}

/// <summary>

/// Функция события для добавления информации в журнал о изменении объектов

/// </summary>

/// <param name="source">Источник данных для обработки</param>

/// <param name="e">Необходимые параметры</param>

public void CollectionReferenceChanged(object source, CollectionHandlerEventArgs e)

{

JournalEntry je = new JournalEntry(e.Name, e.ChangeCollection, e.Object.ToString());

List.Add(je);

}

}

}

# ПРИЛОЖЕНИЕ В **Программа CollectionHandlerEventArgs.cs**

using lab12\_part4;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab

{

/// <summary>

/// Класс, содержащий параметры для повсеместной обработки событий

/// </summary>

public class CollectionHandlerEventArgs : EventArgs

{

public string Name { get; set; } // Информационное поле с названием коллекции

public string ChangeCollection { get; set; }//Информационное поле с операцией, проведённой над коллекцией

public Point Object { get; set; } // Объект, над которым производились измения или добавленный

public CollectionHandlerEventArgs(string name, string changeCollection, Point obj)

{

this.Name = name;

this.ChangeCollection = changeCollection;

this.Object = obj;

}

}

}

# ПРИЛОЖЕНИЕ Г **Программа MyNewCollection.cs**

using lab12\_part4;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.InteropServices.JavaScript;

using System.Text;

using System.Threading.Tasks;

namespace lab

{

// Делегат, собирающий ссылки, играет роль обработчика событий

public delegate void CollectionHandler(object source, CollectionHandlerEventArgs args);

/// <summary>

/// Класс для работы с новой коллекцией

/// </summary>

public class MyNewCollection : MyCollection

{

public Point Root { get; set; } //Информационное поле с корнем дерева

public int Size { get; set; }//Информационное поле с размером коллекции

public MyCollection myCollection {get; set;}// Информационное поле с базовой коллекцией

public string Name { get; set; }

/// <summary>

/// Конструктор для пустой коллекции

/// </summary>

public MyNewCollection()

{

Root = null;

Size = 0;

Count = 0;

Name = "NOName";

}

/// <summary>

/// Конструктор для коллекции с данными

/// </summary>

/// <param name="Name"></param>

/// <param name="size"></param>

public MyNewCollection(string Name, int size)

{

this.Name = Name;

MyCollection temp = new MyCollection(size);

MyCollection.AddFewElementsToCollection(temp, size);

this.Root = temp.Root;

this.Values = temp.Values;

this.myCollection = temp;

this.Size = temp.Size;

}

/// <summary>

/// Список со значениями дерева

/// </summary>

public List<Point> Values

{

get

{

return MakeListValues(Root);

}

set

{

}

}

//События для изменения элементов и добавления элементов в коллекцию

public event CollectionHandler CollectionCountChanged;

public event CollectionHandler CollectionReferenceChanged;

/// <summary>

/// Функция с вызовым события при добавлении или удалении

/// </summary>

/// <param name="source">Коллекция</param>

/// <param name="args">Параметры коллекции</param>

public virtual void OnCollectionCountChanged(object source, CollectionHandlerEventArgs args)

{

CollectionCountChanged?.Invoke(source, args);

}

/// <summary>

/// Функция с вызовым события при измении объекта

/// </summary>

/// <param name="source">Коллекция</param>

/// <param name="args">Параметры коллекции</param>

public virtual void OnCollectionReferenceChanged(object source, CollectionHandlerEventArgs args)

{

CollectionReferenceChanged?.Invoke(source, args);

}

/// <summary>

/// Функция удаления элемента

/// </summary>

/// <param name="position">индекс нужного элемента</param>

/// <returns>Логическое значение</returns>

public bool Remove(int position)

{

if (position <= this.myCollection.Size)

{

OnCollectionCountChanged(this, new CollectionHandlerEventArgs(this.Name, "Удаление", Values[position-1]));

MyCollection myCollection = new MyCollection(this.myCollection);

bool isOk = MyCollection.Remove(ref myCollection, position);

this.myCollection = myCollection;

this.Root = myCollection.Root;

this.Size = myCollection.Size;

return isOk;

}

else

{

return false;

}

}

/// <summary>

/// Функция добавления элемента

/// </summary>

/// <returns>Логическое значение</returns>

public bool Add()

{

Point element = Point.RandomInit();

OnCollectionCountChanged(this, new CollectionHandlerEventArgs(this.Name, "Добавление", element));

MyCollection.AddElementToCollection(this.myCollection, element);

MyCollection.MakeIndex(myCollection);

this.Size = myCollection.Size;

this.Root = myCollection.Root;

return true;

}

/// <summary>

/// Функция изменения объекта по индексу с именем

/// </summary>

/// <param name="index">индекс объекта</param>

/// <param name="name">новое имя объекта</param>

/// <returns>Логическое значение</returns>

public bool Change(int index, string name)

{

Point element = new Point();

MyCollection.ChangeIndex(myCollection, index, name, ref element);

OnCollectionReferenceChanged(this, new CollectionHandlerEventArgs(this.Name, "Изменение объекта", element));

this.Size = myCollection.Size;

this.Root = myCollection.Root;

return true;

}

/// <summary>

/// Вывод коллекции

/// </summary>

/// <param name="collection">Базовая коллекция</param>

public static void Show(MyNewCollection collection)

{

foreach (lab12\_part4.Point item in collection.myCollection)

{

Console.WriteLine(item);

}

}

}

}

# ПРИЛОЖЕНИЕ Д **Код Unit-Тестов**

using lab;

using lab12\_part4;

namespace lab.Tests

{

[TestClass]

public class UnitTest1

{

[TestMethod]

public void TestJournal1()

{

Journal journal = new Journal();

MyNewCollection myCollection = new MyNewCollection("Name", 10);

myCollection.CollectionCountChanged += new CollectionHandler(journal.CollectionCountChanged);

myCollection.Remove(2);

Assert.AreEqual(journal.list.Count, 1);

}

[TestMethod]

public void TestJournal2()

{

Journal journal = new Journal();

MyNewCollection myCollection = new MyNewCollection("Name", 10);

myCollection.CollectionCountChanged += new CollectionHandler(journal.CollectionCountChanged);

myCollection.Add();

Assert.AreEqual(journal.list.Count, 1);

}

[TestMethod]

public void TestJournal3()

{

Journal journal = new Journal();

MyNewCollection myCollection = new MyNewCollection("Name", 10);

myCollection.CollectionReferenceChanged += new CollectionHandler(journal.CollectionReferenceChanged);

myCollection.Change(1, "test");

Assert.AreEqual(journal.list.Count, 1);

}

}

}