|  |
| --- |
| 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 №12**  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)

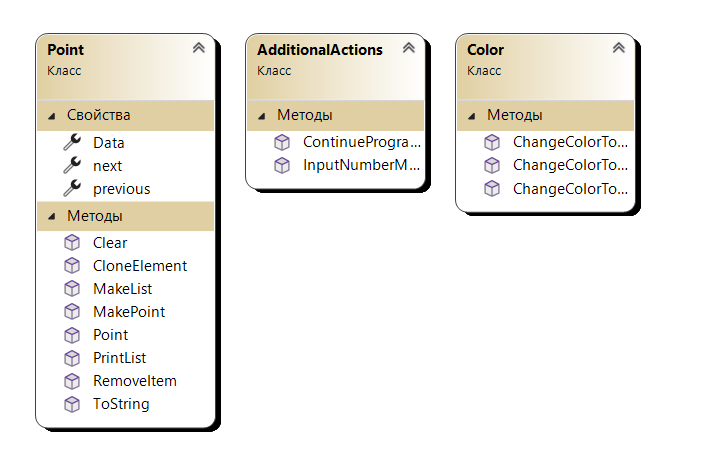
**Часть №1 Двунаправленный список**

**1 Постановка задачи**

1. Сформировать двунаправленный список, в информационное поле записать объекты из иерархии классов лабораторной работы №10.
2. Распечатать полученный список.
3. Выполнить обработку списка в соответствии с заданием.
4. Распечатать полученный список.
5. Выполнить клонирование списка, показать, что под объекты, хранящиеся в информационном поле выделена разная память
6. Удалить список из памяти.

7 вариант – Удалить из списка первый элемент с заданным информационным полем (например, с заданным именем).

**2 Диаграмма классов**



**Рисунок 1 – Диаграмма классов**

Первая часть состоит из трёх классов, а именно:

* Point – в этом классе реализованы все функции для создания, а также изменения двунаправленного списка.
* AdditionalActions – Этот класс реализует дополнительные функции, необходимые для написания более короткого кода в главном теле программы, например, функция InputNumberMenu, которая реализует ввод целого положительного числа с необходимыми условиями.
* Color – Вспомогательный класс, который выполняет функцию смены цвета в выводе консоли, сделан для более удобного смены цвета.

**3 Программа (листинг)**

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

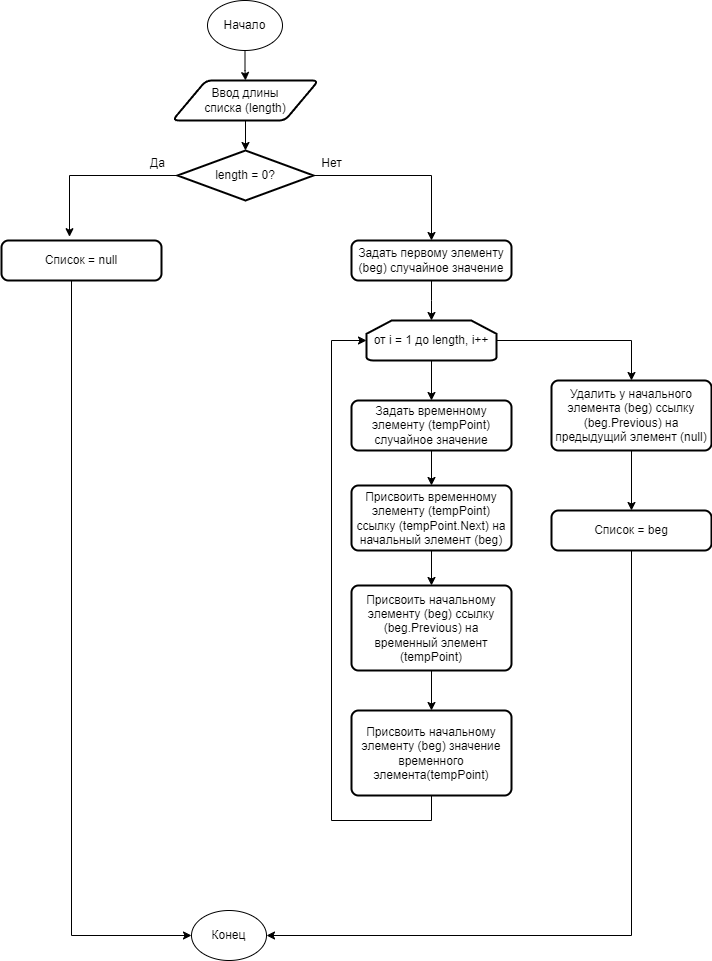
**4 Код Unit-тестов**

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

**5 Анализ покрытия кода тестами**

С помощью Unit-тестов удалось достичь высокий процент покрытия кода, так как в этой работе большое количество методов, основанных на вводе и генерации случайных элементов, значит, что можно исключить их тестирование, ведь нет необходимости перегружать новые функции. Также Unit-тесты покрывают большую область кода. См. Приложение У

**6 Алгоритм построения двунаправленного списка**



**Рисунок 2 – Блок-Схема построения двунаправленного списка**

Блок-схема первой части иллюстрирует последовательность действий для создания двунаправленного списка: создание пустого списка или с заданной длиной. Суть работы алгоритма состоит в том, что значение начального элемента сдвигается в конец, а каждый новый элемент добавляется левее относительно списка.

**Часть №2 Идеально-сбалансированное дерево и дерево поиска**

**1 Постановка задачи**

1. Сформировать идеально сбалансированное бинарное дерево, в информационное поле записать объекты из иерархии классов лабораторной работы №10.

2. Распечатать полученное дерево.

3. Выполнить обработку дерева в соответствии с заданием, вывести полученный результат.

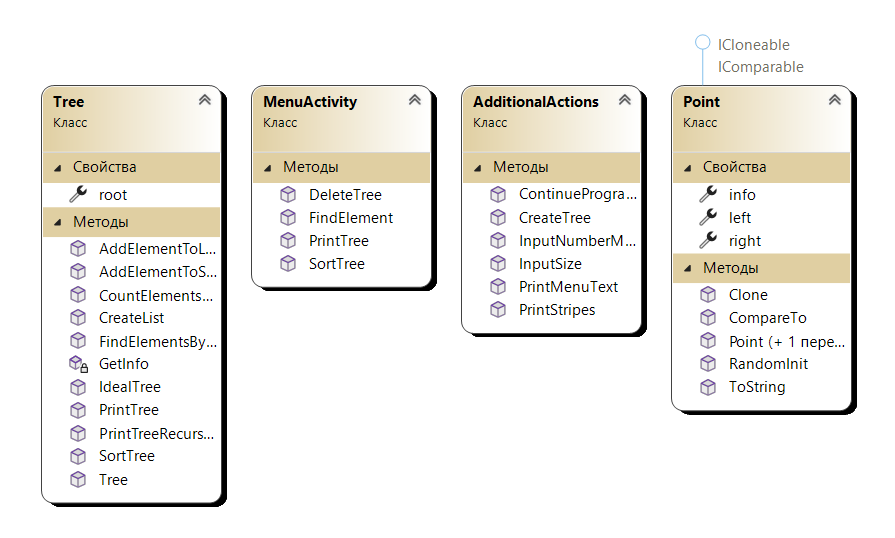
4. Преобразовать идеально сбалансированное дерево в дерево поиска. Информационное поле склонировать.

5. Распечатать полученное дерево.

6. Удалить дерево из памяти.

7 вариант – Найти количество элементов дерева, у которых поле (например, имя) начинается с заданного символа.

**2 Диаграмма классов**



**Рисунок 3 – Диаграмма классов**

Вторая часть состоит из четырёх классов, а именно:

* Point – в этом классе реализованы все функции для создания и работы с элементами дерева: создание, печать, клонирование.
* AdditionalActions – Этот класс реализует дополнительные функции, необходимые для написания более короткого кода в главном теле программы, например, функция InputNumberMenu, которая реализует ввод целого положительного числа с необходимыми условиями.
* Tree –Этот класс реализует функции для работы с деревом : печать, добавление элементов, сортировка и тому подобное.
* MenuActivity – .Вспомогательный класс, который необходим для написания более короткого кода в теле программы, необходим для более удобного написания случаев (case) в одну строчку для switch.

**3 Программа (листинг)**

Код программы представлен в приложении. См. Приложение Д-З

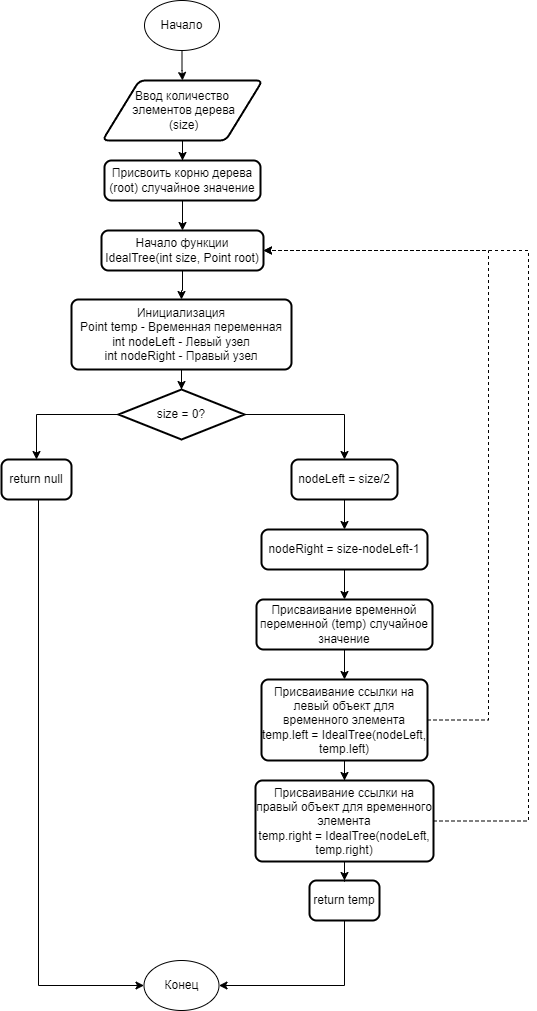
**4 Код Unit-тестов**

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

**5 Анализ покрытия кода тестами**

С помощью Unit-тестов удалось достичь высокий процент покрытия кода, так как в этой работе большое количество методов, основанных на вводе и генерации случайных элементов, значит, что можно исключить их тестирование, ведь нет необходимости перегружать новые функции. Также Unit-тесты покрывают большую область кода. См. Приложение У

**6 Алгоритм построения идеально-сбалансированного дерева**



**Рисунок 4 – Блок-Схема построения идеально-сбалансированного дерева**

Блок-схема второй части демонстрирует работу алгоритма для создания идеально-сбалансированного дерева. Этот процесс основан на рекурсивной функции с созданием дерева слева-направо. Сначала считается кол-во элементов в левом поддереве, затем в правом. А затем происходит присваивание новых элементов.

**Часть №3 Хэш-таблица с открытой адресацией**

**1 Постановка задачи**

1. Создать хеш-таблицу и заполнить ее элементами.

2. Выполнить поиск элемента в хеш-таблице

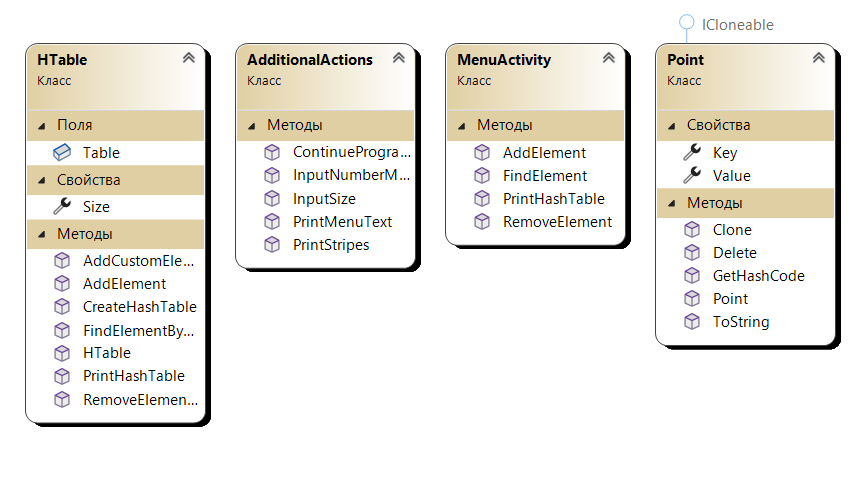
3. Удалить найденный элемент из хеш-таблицы.

4. Выполнить поиск элемента в хеш-таблице

5. Показать, что будет при добавлении элемента в хеш-таблицу, если в таблице уже находится максимальное число элементов (для метода открытой адресации, для метода цепочек просто показать добавление в таблицу).

7 вариант – Открытая адресация, поиск и удаление по ключу.

**2 Диаграмма классов**



**Рисунок 5 – Диаграмма классов**

Третья часть состоит из четырёх классов, а именно:

* Point – в этом классе реализованы все функции для создания и работы с элементами хэш-таблицы: создание, печать, клонирование и удаление.
* AdditionalActions – Этот класс реализует дополнительные функции, необходимые для написания более короткого кода в главном теле программы, например, функция InputNumberMenu, которая реализует ввод целого положительного числа с необходимыми условиями.
* HTable –Этот класс реализует функции для работы с хэш-таблицей: печать, добавление элементов, поиск элементов и тому подобное.
* MenuActivity – .Вспомогательный класс, который необходим для написания более короткого кода в теле программы, необходим для более удобного написания случаев (case) в одну строчку для switch.

**3 Программа (листинг)**

Код программы представлен в приложении. См. Приложение И-М

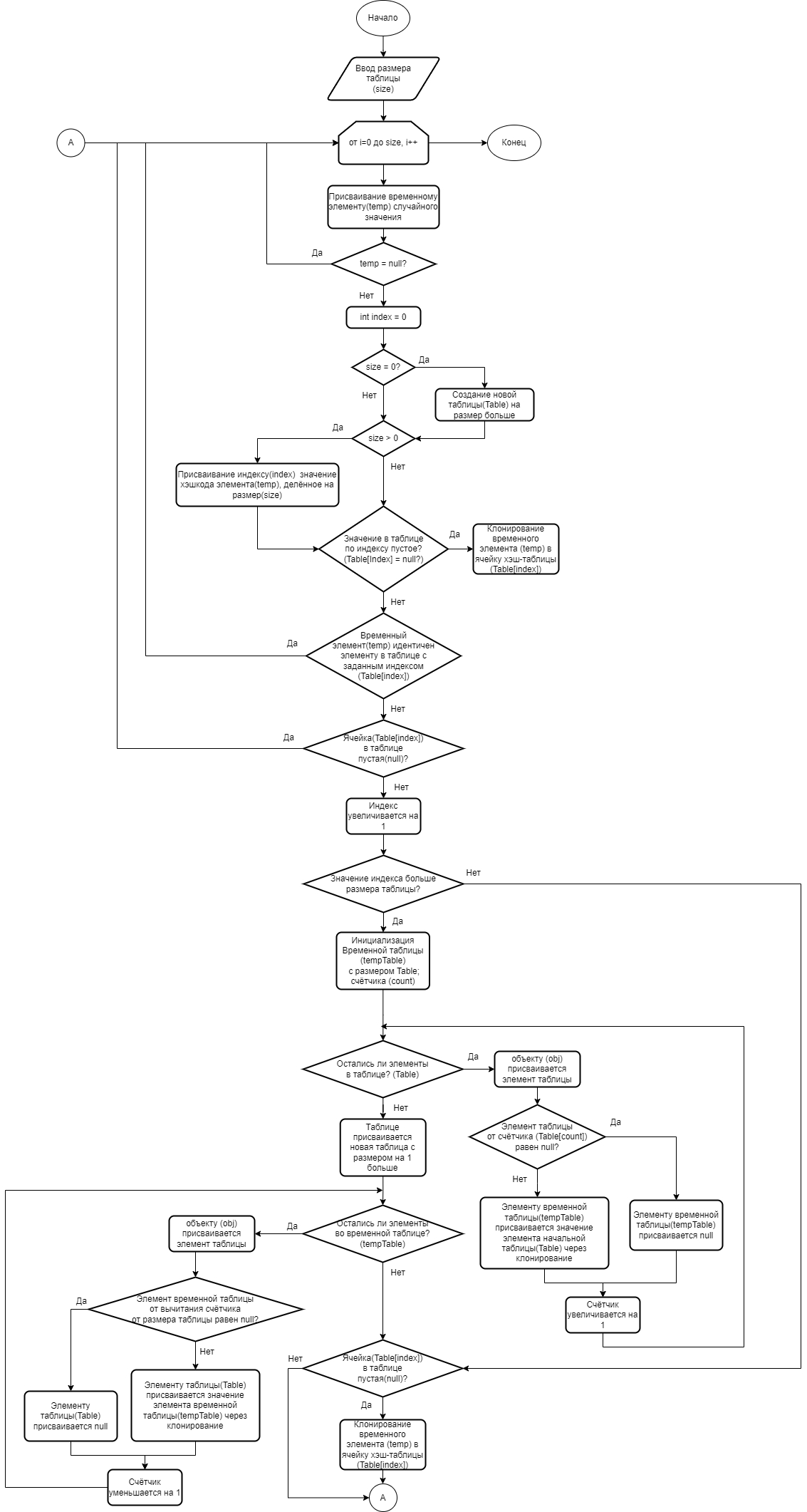
**4 Код Unit-тестов**

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

**5 Анализ покрытия кода тестами**

С помощью Unit-тестов удалось достичь высокий процент покрытия кода, так как в этой работе большое количество методов, основанных на вводе и генерации случайных элементов, значит, что можно исключить их тестирование, ведь нет необходимости перегружать новые функции. Также Unit-тесты покрывают большую область кода. См. Приложение У

**6 Алгоритм построения хэш-таблицы**



**Рисунок 6 – Блок-Схема построения хэш-таблицы**

Блок-схема третьей части демонстрирует работу алгоритма для создания хэш-таблицы с открытой адресацией, из-за того, что при открытой адресации при добавлении элемента на позицию, в которой уже есть элемент сдвигается присваивание, блок-схема содержит трудные в реализации и иллюстрации конструкции foreach, while и т.п. Сначала происходят попытки присвоить элемент в первую свободную следующую ячейку, если не получается в следующую, то продолжается проходка, если заканчивается таблица, то создаётся новая идентичная с размером, большим на 1, с новым последним элементом на конце.

**Часть №4 Реализация коллекции (дерево поиска)**

**1 Постановка задачи**

Реализовать обобщенную коллекцию, указанную в варианте. Для этого:

1. Реализовать конструкторы:

• public MyCollection() - предназначен для создания пустой коллекции.

• public MyCollection (int capacity) - создает пустую коллекцию с начальной емкостью, заданной параметром capacity.

• public MyCollection (MyCollection c) - служит для создания коллекции, которая инициализируется элементами и емкостью коллекции, заданной параметром с.

2. Для всех коллекций реализовать:

• методы для добавления одного или нескольких элементов в коллекцию;

• методы для удаления одного или нескольких элементов из коллекции;

• метод для поиска элемента по ключу и/или по значению;

• метод для глубокого клонирования коллекции (вместе с элементами);

• метод для поверхностного копирования;

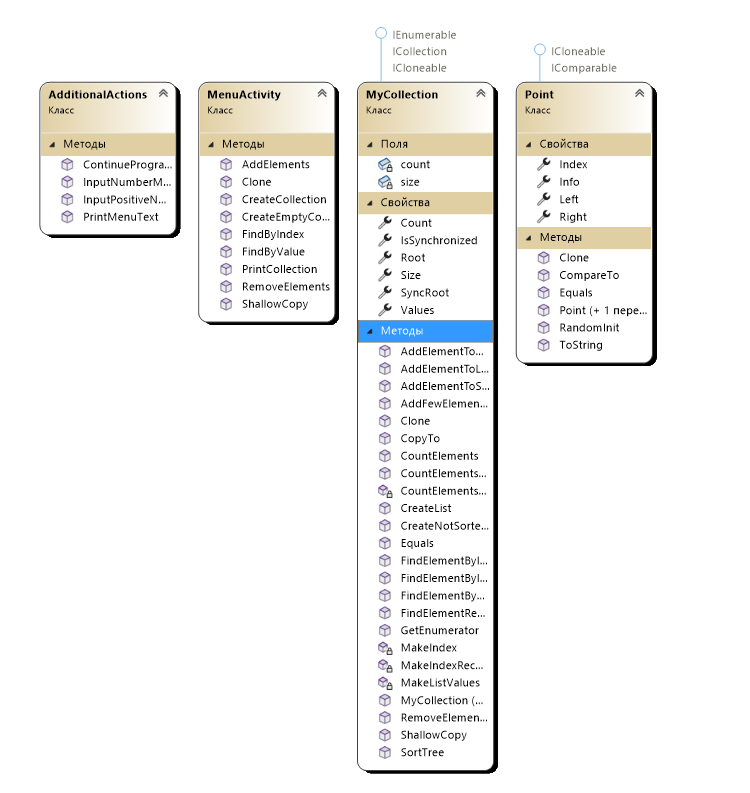
• индексатор для получения/изменения элемента по индексу/ключу.

3. Для всех коллекций реализовать интерфейсы IEnumerable, ICollection.

4. Написать демонстрационную программу, в которой создаются коллекции, и демонстрируется работа всех реализованных методов, в том числе, перебор коллекции циклом foreach.7 вариант – LinkedList<T>, Dictionary<K, T> необходимо создать класс , в котором будут производиться инициализация коллекций, заполнение элементами, нахождение элементов, времени их поиска, а также печать.

7 Вариант (Задание на выбор) – Дерево поиска.

**2 Диаграмма классов**



**Рисунок 7 – Диаграмма классов**

Четвёртая часть состоит из четырёх классов, а именно:

* Point – в этом классе реализованы все функции для создания и работы с элементами коллекции: создание, печать, клонирование и сравнение.
* AdditionalActions – Этот класс реализует дополнительные функции, необходимые для написания более короткого кода в главном теле программы, например, функция InputNumberMenu, которая реализует ввод целого положительного числа с необходимыми условиями.
* MyCollection –Этот класс реализует функции для работы с коллекцией: перебор элементов через foreach, добавление элементов, поиск элементов, индексация, клонирование, поверхностное копирование и тому подобное.
* MenuActivity – .Вспомогательный класс, который необходим для написания более короткого кода в теле программы, необходим для более удобного написания случаев (case) в одну строчку для switch.

**3 Программа (листинг)**

Код программы представлен в приложении. См. Приложение Н-С

**4 Код Unit-тестов**

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

**5 Анализ покрытия кода тестами**

С помощью Unit-тестов удалось достичь высокий процент покрытия кода, так как в этой работе большое количество методов, основанных на вводе и генерации случайных элементов, значит, что можно исключить их тестирование, ведь нет необходимости перегружать новые функции. Также Unit-тесты покрывают большую область кода. См. Приложение У

# ПРИЛОЖЕНИЕ А **Программа Program.cs (часть 1)**

using lab12\_part1;

using System.Diagnostics.CodeAnalysis;

using System.Security.Cryptography;

using System.Security.Principal;

// 7 вариант

namespace lab

{

[ExcludeFromCodeCoverage]

internal class Program

{

static void Main(string[] args)

{

bool isEnd = false;

Console.WriteLine("Введите длину списка:");

int length = AdditionalActions.InputNumberMenu();

//Создание списка c длиной length

Point? list = Point.MakeList(length);

Console.Clear();

do

{

Console.WriteLine(@"[1] - Печать

[2] - Удаление первого элемента заданного типа

[3] - Очистка памяти

[0] - Конец");

int input = AdditionalActions.InputNumberMenu();

Console.Clear();

switch (input)

{

case 1:

//Печать списка

Point.PrintList(list);

AdditionalActions.ContinueProgram();

Console.Clear();

break;

case 2:

//Удаление из списка первого элемента с заданным информационным полем

Console.WriteLine("Введите тип элемента:");

string type = Console.ReadLine();

Point.RemoveItem(ref list, type, ref length);

AdditionalActions.ContinueProgram();

break;

case 3:

// Очистка памяти

Point.Clear(ref list);

//Сборщик мусора

GC.Collect();

AdditionalActions.ContinueProgram();

break;

case 0:

isEnd=true;

break;

default:

Console.Clear();

break;

}

}while(!isEnd);

}

}

}

# ПРИЛОЖЕНИЕ Б **Программа Point.cs (часть 1)**

using System;

using System.Collections.Generic;

using System.Diagnostics.Metrics;

using System.Drawing;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Security.Cryptography.X509Certificates;

using System.Text;

using System.Threading.Tasks;

using lab;

using ClassLibraryLab10;

using System.Diagnostics.CodeAnalysis;

namespace lab

{

[ExcludeFromCodeCoverage]

internal class Point

{

public Toy? Data { get; set; }

public Point? next { get; set; }

public Point? previous { get; set; }

[ExcludeFromCodeCoverage]

public Point(Toy data)

{

this.Data = data;

next=null;

}

[ExcludeFromCodeCoverage]

public override string? ToString()

{

return Data?.ToString();

}

[ExcludeFromCodeCoverage]

public static Point MakePoint()

{

Toy tempToy = new Toy();

tempToy.RandomInit();

Point tempPoint = new Point(tempToy);

return tempPoint;

}

[ExcludeFromCodeCoverage]

public static Point? MakeList(int length)

{

if(length ==0)

{

return null;

}

Point beg = MakePoint();

for (int i=1; i<length; i++)

{

Point tempPoint = MakePoint();

tempPoint.next = beg;

beg.previous = tempPoint;

beg = tempPoint;

}

beg.previous = null;

return beg;

}

[ExcludeFromCodeCoverage]

public static void PrintList(Point beg)

{

if (beg is null)

{

Console.WriteLine("Пустой список! ");

return;

}

Point? tempPoint = beg;

int counter = 0;

while (tempPoint is not null)

{

counter++;

Console.WriteLine($"[{counter}] - {tempPoint} ");

Console.WriteLine( );

tempPoint = tempPoint.next;

}

}

[ExcludeFromCodeCoverage]

public static void RemoveItem(ref Point beg, string itemType, ref int length)

{

if (beg is null)

{

Console.WriteLine("Пустой список!");

}

bool isRemoved = false;

Point tempPoint = beg;

int counter = 1;

do

{

if (string.Compare(itemType, tempPoint?.Data?.ToyType)==0)

{

if (length==1)

{

beg = null;

}

else

{

if (tempPoint.next is null)

{

tempPoint = tempPoint?.previous;

tempPoint.next = null;

}

else

{

if (tempPoint.previous is null)

{

beg = tempPoint.next;

beg.previous=null;

}

else

{

tempPoint = tempPoint.previous;

tempPoint.next = tempPoint?.next?.next;

tempPoint = tempPoint?.next;

tempPoint.previous=tempPoint.previous.previous;

}

}

}

isRemoved=true;

break;

}

else

{

tempPoint = tempPoint?.next;

}

counter++;

} while (counter <=length);

if(isRemoved ==false)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Удаление не было выполнено!");

Console.ForegroundColor = ConsoleColor.White;

}

else

{

length=length-1;

}

}

[ExcludeFromCodeCoverage]

public static Point CloneElement(Point list)

{

Point clone = new Point(list.Data);

return clone;

}

[ExcludeFromCodeCoverage]

public static void Clear(ref Point list)

{

if (list is null)

{

Console.WriteLine("Пустой список!");

}

else

{

list.previous = null;

list.next = null;

list = null;

}

}

}

}

# ПРИЛОЖЕНИЕ В **Программа Color.cs (часть 1)**

using System;

using System.Collections.Generic;

using System.Diagnostics.CodeAnalysis;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab12\_part1

{

[ExcludeFromCodeCoverage]

internal class Color

{

public static void ChangeColorToGreen()

{

Console.ForegroundColor = ConsoleColor.Green;

}

public static void ChangeColorToWhite()

{

Console.ForegroundColor = ConsoleColor.White;

}

public static void ChangeColorToRed()

{

Console.ForegroundColor = ConsoleColor.Red;

}

}

}

# ПРИЛОЖЕНИЕ Г **Программа AdditionalActions.cs (часть 1)**

using System;

using System.Collections.Generic;

using System.Diagnostics.CodeAnalysis;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab12\_part1

{

[ExcludeFromCodeCoverage]

internal class AdditionalActions

{

public static void ContinueProgram()

{

Color.ChangeColorToRed();

Console.WriteLine("Введите enter, чтобы продолжить.");

ConsoleKey key;

do

{

key =Console.ReadKey().Key;

}while (key != ConsoleKey.Enter);

Color.ChangeColorToWhite();

Console.Clear();

}

public static int InputNumberMenu()

{

int enter;

bool isInt = false;

do

{

isInt = int.TryParse(Console.ReadLine(), out enter);

} while (isInt != true);

return enter;

}

}

}

# ПРИЛОЖЕНИЕ Д **Программа Program.cs (часть 2)**

using lab;

using System.Diagnostics.CodeAnalysis;

namespace Part2

// Вариант 7

{

[ExcludeFromCodeCoverage]

internal class Program

{

static void Main(string[] args)

{

Console.WriteLine("Введите количество элементов дерева:");

int size = AdditionalActions.InputSize(); //Размер дерева

Point root = AdditionalActions.CreateTree(size); //Корень дерева с уже созданным деревом по размеру

bool isExit = false;

do // Начало цикла функций меню

{

Console.Clear();

AdditionalActions.PrintMenuText();

int input = AdditionalActions.InputNumberMenu();

Console.Clear();

switch (input)

{

case 1: // Печать дерева

MenuActivity.PrintTree(root, size);

break;

case 2: // Нахождение количества элементов по первому символу в наименовании

MenuActivity.FindElement(root);

break;

case 3: // Создание идеально-сбалансированного дерева поиска на основе уже существующего И-С дерева

MenuActivity.SortTree(root, size);

break;

case 4: // Удаление дерева из памяти

MenuActivity.DeleteTree(ref root);

break;

case 0: // Выход из программы

isExit = true;

break;

default:

break;

}

} while (!isExit); // Конец цикла функций меню

}

}

}

# ПРИЛОЖЕНИЕ Е **Программа Point.cs (часть 2)**

using ClassLibraryLab10;

using lab;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Part2

{

public class Point : ICloneable, IComparable

{

public Toy info { get; set; }//Информационное поле класса Point

public Point left { get; set; } //Поле с ссылкой на левое поддерево класса Point

public Point right { get; set; } //Поле с ссылкой на правое поддерево класса Point

/// <summary>

/// Создание элемента класса Point по данным

/// </summary>

/// <param name="info">Данные класса Toy</param>

public Point(Toy info)

{

this.info = info;

left = null;

right = null;

}

/// <summary>

/// Создание элемента класса Point с пустыми данными

/// </summary>

public Point()

{

left = null;

right = null;

}

/// <summary>

/// Функция создания элемента класса Point со случайными данными

/// </summary>

/// <returns>Заполненный элемент класса Point</returns>

public static Point RandomInit()

{

Toy tempToy = new Toy();

tempToy.RandomInit();

return new Point(tempToy);

}

/// <summary>

/// Функция полного клонирования данных

/// </summary>

/// <returns>Склонированные данные</returns>

public object Clone()

{

return new Point((Toy)info.Clone());

}

/// <summary>

/// Перегрузка функции для вывода Point

/// </summary>

/// <returns>Информационное поле в string</returns>

public override string ToString()

{

return info.ToString();

}

/// <summary>

/// Перегрузка CompareTo для сравнений

/// </summary>

/// <param name="obj">Объект, который будет сравниваться</param>

/// <returns>Результат сравнения по количеству товара</returns>

public int CompareTo(object? obj)

{

if (!(obj is Point))

return -1;

Point point = (Point)obj;

return info.Amount.CompareTo(point.info.Amount);

}

}

}

# ПРИЛОЖЕНИЕ Ё **Программа MenuActivity.cs (часть 2)**

using System;

using System.Collections.Generic;

using System.Diagnostics.CodeAnalysis;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Part2

{

/// <summary>

/// Основное меню.

/// </summary>

[ExcludeFromCodeCoverage]

internal class MenuActivity

{

/// <summary>

/// Фунция основного меню для печати дерева слева-направо

/// </summary>

/// <param name="root">Корень дерева</param>

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

public static void PrintTree(Point root, int size)

{

AdditionalActions.PrintStripes();

Tree.PrintTree(root, size);

AdditionalActions.ContinueProgram();

Console.Clear();

}

/// <summary>

/// Функция основного меню для поиска количества элементов по первой букве названий

/// </summary>

/// <param name="root">Корень дерева</param>

public static void FindElement(Point root)

{

AdditionalActions.PrintStripes();

Tree.FindElementsBySymbol(root);

AdditionalActions.ContinueProgram();

Console.Clear();

}

/// <summary>

/// Функция основного меню для создания идеально-сбалансированного дерева поиска на основе идеально-сбалансированного дерева

/// </summary>

/// <param name="root">Корень дерева</param>

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

public static void SortTree(Point root, int size)

{

AdditionalActions.PrintStripes();

Point sortedRoot = new Point();

List<Point> list = Tree.CreateList(root);

Tree.SortTree(ref sortedRoot, ref list);

Tree.PrintTree(sortedRoot, size);

AdditionalActions.ContinueProgram();

Console.Clear();

}

/// <summary>

/// Функция основного меню для удаления дерева

/// </summary>

/// <param name="root">Корень дерева</param>

public static void DeleteTree(ref Point root)

{

AdditionalActions.PrintStripes();

root = null;

Console.WriteLine("Дерево было удалено.");

AdditionalActions.ContinueProgram();

Console.Clear();

}

}

}

# ПРИЛОЖЕНИЕ Ж **Программа Tree.cs (часть 2)**

using ClassLibraryLab10;

using lab;

using System;

using System.Collections.Generic;

using System.ComponentModel.Design.Serialization;

using System.Linq;

using System.Net.Http.Headers;

using System.Security.AccessControl;

using System.Text;

using System.Threading.Tasks;

namespace Part2

{

public class Tree

{

public Point root { get; set; }//Информационное поле корня

/// <summary>

/// Создание деревва с пустым информационным полем

/// </summary>

public Tree()

{

root = null;

}

/// <summary>

/// Вспомогательная функция получения случайных данных для создания дерева

/// </summary>

/// <returns>Возвращает данные класса Toy</returns>

static Toy GetInfo()

{

Toy info = new Toy();

info.RandomInit();

return info;

}

/// <summary>

/// Функция создания идеально-сбалансированного дерева

/// </summary>

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

/// <param name="root">Корень дерева</param>

/// <returns>Возвращает корень И-С дерева с элементами</returns>

public static Point IdealTree(int size, Point root)

{

Point temp; //Временная переменная, которая в конце будет возвращена

int nodeLeft; //Левый узел

int nodeRight; //Правый узел

if (size==0) //Если размер равен нулю, то дерево пустое

{

root = null;

return root;

}

nodeLeft = size/2; //Количество левых узлов

nodeRight = size-nodeLeft-1; //Количество правых узлов

Toy info = GetInfo();

temp = new Point(info); //Создание нового элемента

temp.left = IdealTree(nodeLeft, temp.left); //Рекурсия для левого поддерева

temp.right = IdealTree(nodeRight, temp.right); //Рекурсия для правого поддерева

return temp;

}

/// <summary>

/// Функция печати дерева слева-направо

/// </summary>

/// <param name="root">Корень дерева</param>

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

public static void PrintTree(Point root, int size)

{

int rootSize = size; // Переменная размера дерева, чтобы "подчеркнуть" корень

PrintTreeRecursion(root, size, rootSize);

}

/// <summary>

/// Вспомогательная функция для печати дерева, которая выполняет рекурсию

/// </summary>

/// <param name="root">Корень дерева</param>

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

/// <param name="rootSize">Размер всего дерева, который не будет меняться</param>

public static void PrintTreeRecursion(Point root, int size, int rootSize)

{

if (root != null)

{

PrintTreeRecursion(root.left, size+1, rootSize);

int counter = 0;

for (int i = 0; i < size-1; i++)

{

Console.Write("-");

counter++;

}

if (counter == rootSize-1)//Если счётчик равен размеру всего дерева, то выделяется

{

Console.BackgroundColor = ConsoleColor.White;

Console.ForegroundColor = ConsoleColor.Black;

}

Console.WriteLine(root.info);

Console.BackgroundColor = ConsoleColor.Black;

Console.ForegroundColor = ConsoleColor.White;

Console.WriteLine();

PrintTreeRecursion(root.right, size+1, rootSize);

}

}

/// <summary>

/// Вспомогательная функция подсчёта элементов для нахождения по символу, в котороый выполняется основная рекурсия

/// </summary>

/// <param name="root">Корень дерева</param>

/// <param name="symbol">Символ, по которому идёт поиск</param>

/// <param name="counter">Счётчик</param>

public static void CountElementsBySymbol(Point root, string symbol, ref int counter)

{

if (root != null)

{

CountElementsBySymbol(root.left, symbol, ref counter);

if (string.Compare(Char.ToString(root.info.Name[0]), symbol) == 0)

{

counter++;

}

CountElementsBySymbol(root.right, symbol, ref counter);

}

}

/// <summary>

/// Основная функция поиска количества элементов по первому символу в названии

/// </summary>

/// <param name="root">Корень дерева</param>

public static void FindElementsBySymbol(Point root)

{

string symbol;

do //Цикл, который определяет единственный непустой символ, по которому будет производиться поиск

{

Console.WriteLine("Введите символ, по которому будут найдено количество элементов, начинающегося с него:");

symbol = Console.ReadLine();

Console.Clear();

} while (symbol.Length != 1);

int counter = 0;

CountElementsBySymbol(root, symbol, ref counter);

Console.WriteLine($"Количество элементов, начинающихся с символа '{symbol}' - {counter}");

}

/// <summary>

/// Вспомогательная функция для создания дерева поиска, которая добавляет элемент в дерево

/// </summary>

/// <param name="sortedRoot"></param>

/// <param name="element"></param>

/// <returns></returns>

public static Point AddElementToSortTree(ref Point sortedRoot, Point element)

{

if (element != null)

{

Point temp = sortedRoot;

Point temp2 = null;

bool ok = false;

while (temp != null && !ok)

{

temp2 = temp;

if (element.info.Amount == temp.info.Amount && element.info.Price == temp.info.Price) //Если элемент является корнем

ok = true;

else // Если элемент меньше, то уходит в левое поддерево

if ((element.info.Amount < temp.info.Amount) || (element.info.Amount == temp.info.Amount &&element.info.Price < temp.info.Price))

temp = temp.left;

else

temp = temp.right;

}

if (ok)

return temp;

if (element.info.Amount < temp2.info.Amount)

temp2.left = (Point)element.Clone();

else

temp2.right = (Point)element.Clone();

return element;

}

else

return null;

}

/// <summary>

/// Основная функция сортировки дерева

/// </summary>

/// <param name="sortedRoot">Корень нового дерева</param>

/// <param name="list">Список склонированных элементов без ссылок старого дерева</param>

public static void SortTree(ref Point sortedRoot, ref List<Point> list)

{

if (list != null)

{

sortedRoot = list[list.Count/2];

list.RemoveAt(list.Count/2);

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

{

AddElementToSortTree(ref sortedRoot, list[i]);

}

}

else

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Невозможно создать дерево поиска!");

Console.ForegroundColor = ConsoleColor.White;

}

}

/// <summary>

/// Вспомогательная функция, добавляющая элемент в список с полным клонированием

/// </summary>

/// <param name="root">Корень дерева</param>

/// <param name="list">Список с элементами</param>

public static void AddElementToList(Point root, ref List<Point> list)

{

if (root != null)

{

AddElementToList(root.left, ref list);

list.Add((Point)root.Clone());

AddElementToList(root.right, ref list);

}

}

/// <summary>

/// Всопомгательная функция для создания дерева поиска, которая создаёт список элементов без ссылок из старого дерева

/// </summary>

/// <param name="root">Корень старого дерева</param>

/// <returns>Бессылочный список с элементами старого дерева</returns>

public static List<Point> CreateList(Point root)

{

List<Point> list = new List<Point>();

if (root != null)

{

AddElementToList(root, ref list);

list.Sort();

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

{

list[i].left=null;

list[i].right=null;

}

return list;

}

else

{

return null;

}

}

}

}

# ПРИЛОЖЕНИЕ З **Программа AdditionalActions.cs (часть 2)**

using System;

using System.Collections.Generic;

using System.Diagnostics.CodeAnalysis;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Part2

{

[ExcludeFromCodeCoverage]

internal class AdditionalActions

{

/// <summary>

/// Ввод размера дерева

/// </summary>

/// <returns>Целочисленный размер</returns>

public static int InputSize()

{

int size;

bool isNumber = false;

// Начало цикла ввода размера

do

{

isNumber = int.TryParse(Console.ReadLine(), out size);

if (size<0) // Если размер, который не может существовать

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Размер дерева не может быть меньше нуля!");

Console.ForegroundColor = ConsoleColor.White;

}

} while(!(isNumber && size>=0));

// Конец цикла ввода размера

return size;

}

/// <summary>

/// Функция ожидания нажатия клавиши "Enter" для продолжения программы

/// </summary>

public static void ContinueProgram()

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Введите enter, чтобы продолжить.");

ConsoleKey key;

do

{

key =Console.ReadKey().Key;

} while (key != ConsoleKey.Enter);

Console.ForegroundColor = ConsoleColor.White;

Console.Clear();

}

/// <summary>

/// Функция ввода целого числа для выбора пунктов меню

/// </summary>

/// <returns>Целое число</returns>

public static int InputNumberMenu()

{

int enter;

bool isInt = false;

do

{

isInt = int.TryParse(Console.ReadLine(), out enter);

} while (isInt != true);

return enter;

}

/// <summary>

/// Функция вывода разделительных полос (пришлось добавить из-за багов нового терминала в Win 11)

/// </summary>

public static void PrintStripes()

{

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("- - - - - - - - - - - - - -");

Console.ForegroundColor = ConsoleColor.White;

}

/// <summary>

/// Функция вывода контекстного меню

/// </summary>

public static void PrintMenuText()

{

Console.WriteLine(" - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -");

Console.WriteLine(@$"| Меню: |

| |

| [1] - Распечатать дерево. |

| [2] - Найти количество элементов, начинающихся с определённой буквы. |

| [3] - Отсортировать дерево (Переделать в дерево поиска) и распечатать его. |

| [4] - Удалить из памяти. |

| |

| [0] - Выход. |");

Console.WriteLine(" - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -");

}

/// <summary>

/// Функция создания идеально-сбалансированного дерева

/// </summary>

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

/// <returns>Корень уже готового дерева</returns>

public static Point CreateTree(int size)

{

Point root = new Point();

root = Point.RandomInit();

root = Tree.IdealTree(size, root);

Console.Clear();

return root;

}

}

}

# ПРИЛОЖЕНИЕ И **Программа Program.cs (часть 3)**

using ClassLibraryLab10;

using System.Diagnostics.CodeAnalysis;

namespace Lab12\_part3

{

// Вариант 7

[ExcludeFromCodeCoverage]

internal class Program

{

static void Main(string[] args)

{

Console.WriteLine("Введите размер хэштаблицы:");

int size = AdditionalActions.InputSize(); //Размер дерева

HTable hashTable = new HTable(size);

HTable.CreateHashTable(ref hashTable, size);

bool isExit = false;

do // Начало цикла функций меню

{

Console.Clear();

AdditionalActions.PrintMenuText();

int input = AdditionalActions.InputNumberMenu();

Console.Clear();

switch (input)

{

case 1: // Печать таблицы

MenuActivity.PrintHashTable(hashTable, size);

break;

case 2: // Нахождение элемент по ключу

MenuActivity.FindElement(hashTable);

break;

case 3: // Удаление элемента из хэштаблицы

MenuActivity.RemoveElement(hashTable, size);

break;

case 4: // Добавить элемент в хэштаблицу

MenuActivity.AddElement(ref hashTable);

break;

case 0: // Выход из программы

isExit = true;

break;

default:

break;

}

} while (!isExit); // Конец цикла функций меню

}

}

}

# ПРИЛОЖЕНИЕ Й **Программа Point.cs (часть 3)**

using ClassLibraryLab10;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading.Tasks;

namespace Lab12\_part3

{

public class Point : ICloneable

{

public Toy Value { get; set; }

public int Key { get; set; }

public Point (Toy value)

{

Value = value;

Key = value.Amount;

}

public override string ToString()

{

return $"[{Key}] : {Value}";

}

public override int GetHashCode()

{

int hashCode = 0;

int tempValue = Value.Amount;

while (tempValue>0)

{

hashCode += tempValue % 10;

tempValue = tempValue / 10;

}

return hashCode;

}

public static Point Delete()

{

return null;

}

public object Clone()

{

return new Point((Toy)Value.Clone());

}

}

}

# ПРИЛОЖЕНИЕ К **Программа MenuActivity.cs (часть 3)**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Diagnostics.CodeAnalysis;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab12\_part3

{

[ExcludeFromCodeCoverage]

internal class MenuActivity

{

public static void PrintHashTable(HTable hashTable, int size)

{

AdditionalActions.PrintStripes();

hashTable.PrintHashTable();

AdditionalActions.ContinueProgram();

Console.Clear();

}

public static void FindElement(HTable hashTable)

{

AdditionalActions.PrintStripes();

Console.WriteLine("Введите ключ, по которому будет найден элемент:");

int key = AdditionalActions.InputNumberMenu();

hashTable.FindElementByKey(key);

AdditionalActions.ContinueProgram();

Console.Clear();

}

public static void RemoveElement(HTable hashTable, int size)

{

AdditionalActions.PrintStripes();

Console.WriteLine("Введите ключ, по которому должен будет удалён элемент:");

int key = AdditionalActions.InputNumberMenu();

hashTable.RemoveElementByKey(key);

AdditionalActions.ContinueProgram();

Console.Clear();

}

public static void AddElement(ref HTable hashTable)

{

AdditionalActions.PrintStripes();

Console.WriteLine("Введите ключ, С которым будет добавлен новый элемент:");

int key = AdditionalActions.InputNumberMenu();

hashTable.AddCustomElement(ref hashTable, key);

AdditionalActions.ContinueProgram();

Console.Clear();

}

}

}

# ПРИЛОЖЕНИЕ Л **Программа HTable.cs (часть 3)**

using ClassLibraryLab10;

using System;

using System.Collections;

using System.Collections.Generic;

using System.Diagnostics.CodeAnalysis;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Text.Json.Serialization.Metadata;

using System.Threading.Tasks;

namespace Lab12\_part3

{

public class HTable

{

public Point[] Table;

public int Size { get; set; }

public HTable(int size)

{

if (Size >=0)

{

Size = size;

Table = new Point[Size];

}

}

[ExcludeFromCodeCoverage]

public static void CreateHashTable(ref HTable hashTable, int size)

{

Toy temp = new Toy();

for (int i = 0; i < size; i++)

{

temp.RandomInit();

hashTable.AddElement(ref hashTable, temp, size);

}

}

[ExcludeFromCodeCoverage]

public bool AddElement(ref HTable hashTable, Toy element, int size)

{

Point temp = new Point(element);

if (element is null)

return false;

int index = 0;

if (size == 0)

{

Table = new Point[index+1];

}

if (size > 0)

{

index = temp.GetHashCode() % size;

}

if (Table[index] == null)

{

Table[index] = (Point)temp.Clone();

}

else

{

if (string.Compare(temp.ToString(), Table[index].ToString()) == 0)

{

return false;

}

while (Table[index] != null)

{

index += 1;

if (index >= Table.Length)

{

Point[] tempTable = new Point[Table.Length];

int count = 0;

foreach (object obj in Table)

{

if (Table[count] == null)

tempTable[count] = null;

else

tempTable[count] = (Point)Table[count].Clone();

count++;

}

Table = new Point[index+1];

foreach (object obj in tempTable)

{

if (tempTable[tempTable.Length-count] == null)

Table[tempTable.Length-count] = null;

else

Table[tempTable.Length-count] = (Point)tempTable[tempTable.Length-count].Clone();

count--;

}

}

if (Table[index] == null)

{

Table[index] = (Point)temp.Clone();

break;

}

}

}

return true;

}

[ExcludeFromCodeCoverage]

public void PrintHashTable()

{

if (Table == null || Size == 0)

{

Console.WriteLine("Таблица пустая!");

}

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

{

if (Table[i] == null)

{

Console.WriteLine(i + " : ");

}

else

{

Console.Write(i + " : ");

Point temp = Table[i];

Console.WriteLine(temp.ToString());

}

}

}

[ExcludeFromCodeCoverage]

public void RemoveElementByKey(int key)

{

for (int index = 0; index < Table.Length; index++)

{

if (Table[index]!= null)

{

if (Table[index].Key == key)

{

Console.WriteLine();

Console.WriteLine($"Удалён элемент - " +

$"{Table[index]}");

Table[index] = Point.Delete();

break;

}

}

}

if (Table.Count()==0)

{

Table = null;

Size = 0;

}

}

[ExcludeFromCodeCoverage]

public void FindElementByKey(int key)

{

if (Table.Length != 0)

{

for (int index = 0; index < Table.Length; index++)

{

if (Table[index]!= null)

{

if (Table[index].Key == key)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine(Table[index]);

Console.ForegroundColor = ConsoleColor.White;

break;

}

}

}

}

}

[ExcludeFromCodeCoverage]

public void AddCustomElement(ref HTable hashTable, int key)

{

Toy tempToy = new Toy();

tempToy.RandomInit();

tempToy.Amount = key;

int size;

if (hashTable.Table == null)

{

size = 0;

}

else

{

size = hashTable.Table.Count();

}

AddElement(ref hashTable, tempToy, size);

}

}

}

# ПРИЛОЖЕНИЕ М **Программа AdditionalActions.cs (часть 3)**

using System;

using System.Collections.Generic;

using System.Diagnostics.CodeAnalysis;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab12\_part3

{

[ExcludeFromCodeCoverage]

internal class AdditionalActions

{

/// <summary>

/// Ввод размера хэштаблицы

/// </summary>

/// <returns>Целочисленный размер</returns>

public static int InputSize()

{

int size;

bool isNumber = false;

// Начало цикла ввода размера

do

{

isNumber = int.TryParse(Console.ReadLine(), out size);

if (size<0) // Если размер, который не может существовать

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Размер таблицы не может быть меньше нуля!");

Console.ForegroundColor = ConsoleColor.White;

}

} while (!(isNumber && size>=0));

// Конец цикла ввода размера

return size;

}

/// <summary>

/// Функция ожидания нажатия клавиши "Enter" для продолжения программы

/// </summary>

public static void ContinueProgram()

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Введите enter, чтобы продолжить.");

ConsoleKey key;

do

{

key =Console.ReadKey().Key;

} while (key != ConsoleKey.Enter);

Console.ForegroundColor = ConsoleColor.White;

Console.Clear();

}

/// <summary>

/// Функция вывода контекстного меню

/// </summary>

public static void PrintMenuText()

{

Console.WriteLine(" - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -");

Console.WriteLine(@$"| Меню: |

| |

| [1] - Распечатать хэштаблицу. |

| [2] - Найти элемент по ключу. |

| [3] - Удалить элемент по ключу. |

| [4] - Добавить элемент с определённым ключом. |

| |

| [0] - Выход. |");

Console.WriteLine(" - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -");

}

/// <summary>

/// Функция ввода целого числа для выбора пунктов меню

/// </summary>

/// <returns>Целое число</returns>

public static int InputNumberMenu()

{

int enter;

bool isInt = false;

do

{

isInt = int.TryParse(Console.ReadLine(), out enter);

} while (isInt != true);

return enter;

}

public static void PrintStripes()

{

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine("- - - - - - - - - - - - - -");

Console.ForegroundColor = ConsoleColor.White;

}

}

}

# ПРИЛОЖЕНИЕ Н **Программа Program.cs (часть 4)**

using ClassLibraryLab10;

using System.Diagnostics.CodeAnalysis;

using System.Threading.Channels;

namespace lab12\_part4

{

[ExcludeFromCodeCoverage]

internal class Program

{

/// <summary>

/// Главный класс программы

/// </summary>

/// <param name="args"> аргументы </param>

static void Main(string[] args)

{

bool isExit = false;

MyCollection collection = new MyCollection();

do // Начало цикла функций меню

{

Console.Clear();

AdditionalActions.PrintMenuText();

int input = AdditionalActions.InputNumberMenu();

Console.Clear();

switch (input)

{

case 0:

isExit = true;

break;

case 1:

collection = MenuActivity.CreateEmptyCollection();

break;

case 2:

collection = MenuActivity.CreateCollection();

break;

case 3:

collection = MenuActivity.AddElements(collection);

break;

case 4:

collection = MenuActivity.RemoveElements(ref collection);

break;

case 5:

MenuActivity.FindByValue(collection);

break;

case 6:

MenuActivity.Clone(collection);

break;

case 7:

MenuActivity.ShallowCopy(collection);

break;

case 8:

MenuActivity.FindByIndex(collection);

break;

case 9:

MenuActivity.PrintCollection(collection);

break;

case 99:

Console.WriteLine(collection.Root);

AdditionalActions.ContinueProgram();

break;

}

} while (!isExit);

}

}

}

# ПРИЛОЖЕНИЕ О **Программа Point.cs (часть 4)**

using ClassLibraryLab10;

using System;

using System.Collections.Generic;

using System.Diagnostics.CodeAnalysis;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab12\_part4

{

public class Point : ICloneable, IComparable

{

public Toy Info { get; set; }//Информационное поле класса Point

public int Index { get; set; }

public Point Left { get; set; } //Поле с ссылкой на левое поддерево класса Point

public Point Right { get; set; } //Поле с ссылкой на правое поддерево класса Point

/// <summary>

/// Создание элемента класса Point по данным

/// </summary>

/// <param name="info">Данные класса Toy</param>

public Point(Toy info, int Index)

{

this.Info = info;

Left = null;

Right = null;

}

/// <summary>

/// Создание элемента класса Point с пустыми данными

/// </summary>

[ExcludeFromCodeCoverage]

public Point()

{

Left = null;

Right = null;

}

/// <summary>

/// Функция создания элемента класса Point со случайными данными

/// </summary>

/// <returns>Заполненный элемент класса Point</returns>

[ExcludeFromCodeCoverage]

public static Point RandomInit()

{

Toy tempToy = new Toy();

tempToy.RandomInit();

return new Point(tempToy, 0);

}

/// <summary>

/// Функция полного клонирования данных

/// </summary>

/// <returns>Склонированные данные</returns>

public object Clone()

{

return new Point((Toy)Info.Clone(), Index);

}

/// <summary>

/// Перегрузка функции для вывода Point

/// </summary>

/// <returns>Информационное поле в string</returns>

[ExcludeFromCodeCoverage]

public override string ToString()

{

return "["+Index+"] "+Info.ToString();

}

/// <summary>

/// Перегрузка CompareTo для сравнений

/// </summary>

/// <param name="obj">Объект, который будет сравниваться</param>

/// <returns>Результат сравнения по количеству товара</returns>

public int CompareTo(object? obj)

{

if (!(obj is Point))

return -1;

Point point = (Point)obj;

return Info.Amount.CompareTo(point.Info.Amount);

}

public override bool Equals(object? obj)

{

if (obj is Point point)

{

return ((this.Index == point.Index)&&(this.Info == point.Info));

}

else

{

return false;

}

}

}

}

# ПРИЛОЖЕНИЕ П **Программа MyCollecyion.cs (часть 4)**

using ClassLibraryLab10;

using System.Collections;

using System.Diagnostics.CodeAnalysis;

//Дерево поиска

namespace lab12\_part4

{

/// <summary>

/// Класс, в котором реализуется коллекция и её функции

/// </summary>

public class MyCollection : IEnumerable, ICollection, ICloneable

{

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

int size = 0;

int count = 0;

public int Size // Информационное поле с размером коллекции

{

set

{

size = value;

}

get

{

return size;

}

}

public int Count { get; set; } //Информационное поле с количесвтом элементов в коллекции

[ExcludeFromCodeCoverage]

public bool IsSynchronized {

get

{

if (Root == null)

{

return false;

}

else

{

return true;

}

}

set

{

return;

}

}

public List<Point> Values // Информационное поле с элементами в виде массива

{

get

{

return MakeListValues(Root);

}

}

public object SyncRoot { get; set; }

public MyCollection() // Создание пустой коллекции

{

Root = null;

Size = 0;

Count = 0;

}

public MyCollection(int capacity) // Создание пустой коллекции с заданным размером

{

Root = null;

Size = capacity;

Count = 0;

}

public MyCollection(MyCollection collection) // Создание коллекции с элементами из старой коллекции

{

Root = collection.Root;

Size = collection.Size;

Count = collection.Count;

MyCollection myCollection = new MyCollection(Size);

List<Point> list = MyCollection.CreateList(collection.Root);

Point root = collection.Root;

MyCollection.SortTree(ref root, ref list);

collection.Root = root;

MakeIndex(collection);

CountElements(collection);

}

/// <summary>

///Добавление элементов в отсортированное дерево

/// </summary>

/// <param name="sortedRoot">отсортированный корень</param>

/// <param name="element">Элемент</param>

/// <returns>Возвращает элемент</returns>

public static Point AddElementToSortTree(ref Point sortedRoot, Point element)

{

if (element != null)

{

Point temp = sortedRoot;

Point temp2 = null;

bool ok = false;

while (temp != null && !ok)

{

temp2 = temp;

if (element.Info.Amount == temp.Info.Amount && element.Info.Price == temp.Info.Price) //Если элемент является корнем

ok = true;

else // Если элемент меньше, то уходит в левое поддерево

if ((element.Info.Amount < temp.Info.Amount) || (element.Info.Amount == temp.Info.Amount &&element.Info.Price < temp.Info.Price))

temp = temp.Left;

else

temp = temp.Right;

}

if (ok)

return temp;

if (element.Info.Amount < temp2.Info.Amount)

temp2.Left = (Point)element.Clone();

else

temp2.Right = (Point)element.Clone();

return element;

}

else

return null;

}

/// <summary>

///Сортировка дерева

/// </summary>

/// <param name="root">корень дерева</param>

/// <param name="list">список с элементами дерева</param>

public static void SortTree(ref Point root, ref List<Point> list)

{

if (list != null)

{

root = list[list.Count/2];

list.RemoveAt(list.Count/2);

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

{

AddElementToSortTree(ref root, list[i]);

}

}

}

/// <summary>

///Вспомогательная рекурсивная функция, добавляющая элементы коллекции в список

/// </summary>

/// <param name="root">корень</param>

/// <param name="list">список</param>

public static void AddElementToList(Point root, ref List<Point> list)

{

if (root != null)

{

AddElementToList(root.Left, ref list);

list.Add((Point)root.Clone());

AddElementToList(root.Right, ref list);

}

}

/// <summary>

/// Функция, создающая список с элементами коллекции

/// </summary>

/// <param name="root">корень дерева</param>

/// <returns>Возвращает список с элементами</returns>

public static List<Point> CreateList(Point root)

{

List<Point> list = new List<Point>();

if (root != null)

{

AddElementToList(root, ref list);

list.Sort();

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

{

list[i].Left = null;

list[i].Right = null;

}

return list;

}

else

{

return null;

}

}

/// <summary>

/// Функция, добавляющая 1 элемент в коллекцию, является вспомогательной к функции добавления нескольких элементов

/// </summary>

/// <param name="collection">Коллекция, куда будет добавляться элемент</param>

/// <param name="element">Новый элемент</param>

public static void AddElementToCollection(MyCollection collection, Point element)

{

if (collection.Root != null)

{

List<Point> list = MyCollection.CreateList(collection.Root);

list.Add(element);

Point root = collection.Root;

MyCollection.SortTree(ref root, ref list);

collection.Root = root;

}

else

{

collection.Root = element;

}

}

/// <summary>

/// Функция, добавляющая несколько элементов в коллекцию по количеству quantity

/// </summary>

/// <param name="collection">Коллекция, куда будут добавляться элементы</param>

/// <param name="quantity">Количество новых элементов</param>

public static void AddFewElementsToCollection (MyCollection collection, int quantity)

{

for (int i = 0; i < quantity; i++)

{

Toy tempToy = new Toy();

tempToy.RandomInit();

Point temp = new Point(tempToy, 0);

AddElementToCollection(collection, temp);

}

MyCollection.MakeIndex(collection);

collection.Count+=quantity;

List<Point> list = MyCollection.CreateList(collection.Root);

Point root = collection.Root;

MyCollection.SortTree(ref root, ref list);

}

/// <summary>

/// Вспомогательная функция, которая необходима для удаления элементов по первому символу

/// </summary>

/// <param name="rootOrigin">Оригинальный корень</param>

/// <param name="root">элемент в данный момент (для рекурсии)</param>

/// <param name="symbol">Символ, по которому будет происходить удаление</param>

/// <param name="list">Список оставшихся элементов</param>

public static void CountElementsBySymbol(Point rootOrigin, ref Point root, string symbol, ref List<Point> list)

{

if (root != null && rootOrigin!= null)

{

CountElementsBySymbol(rootOrigin.Left ,ref root, symbol, ref list);

if (string.Compare(Char.ToString(rootOrigin.Info.Name[0]), symbol) != 0)

{

list.Add((Point)rootOrigin.Clone());

}

CountElementsBySymbol(rootOrigin.Right, ref root, symbol, ref list);

}

}

/// <summary>

/// Вспомогательная рекурсивная функция для подсчёта общего количества элементов в коллекции

/// </summary>

/// <param name="root">Корень коллекции</param>

/// <param name="index">Количество элементов</param>

static void CountElementsRecursion(Point root, ref int index)

{

if (root != null)

{

CountElementsRecursion(root.Left, ref index);

index++;

CountElementsRecursion(root.Right, ref index);

}

}

/// <summary>

/// Функция подсчёта количества элементов в коллекции

/// </summary>

/// <param name="collection">Коллекция, в которой будет происходить подсчёт</param>

public static void CountElements(MyCollection collection)

{

if (collection.Root != null)

{

int index = 0;

MyCollection.CountElementsRecursion(collection.Root, ref index);

collection.Count = index;

}

}

/// <summary>

/// Функция удаления элементов из коллекции по первому символу названия

/// </summary>

/// <param name="collection">коллекция, в которой будет проведено удаление</param>

[ExcludeFromCodeCoverage]

public static void RemoveElementFromCollectionBySymbol (ref MyCollection collection)

{

if (collection.Root != null)

{

string symbol;

do //Цикл, который определяет единственный непустой символ, по которому будет производиться поиск

{

Console.WriteLine("Введите символ, по которому будут удалены элементы:");

symbol = Console.ReadLine();

Console.Clear();

} while (symbol.Length != 1);

List<Point> list = new List<Point>();

Point root = collection.Root;

CountElementsBySymbol(collection.Root, ref root, symbol, ref list);

collection.Root = root;

MyCollection tempCollection = new MyCollection(list.Count);

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

{

AddElementToCollection(tempCollection, list[i]);

}

collection = new MyCollection(tempCollection);

MyCollection.MakeIndex(collection);

MyCollection.CountElements(collection);

}

}

/// <summary>

/// Вспомогательная рекурсивная функция для поиска элемента по значению количества товара

/// </summary>

/// <param name="root">Корень дерева</param>

/// <param name="amount">Значение количества товара</param>

[ExcludeFromCodeCoverage]

public static void FindElementRecursion(Point root, int amount)

{

if (root != null)

{

FindElementRecursion(root.Left, amount);

if (root.Info.Amount==amount)

{

Console.WriteLine(root);

}

FindElementRecursion(root.Right, amount);

}

}

/// <summary>

/// Функция нахождения элемента по значению количества товара

/// </summary>

/// <param name="collection">коллекция, где будет проведён поиск</param>

/// <param name="amount">Значение количества товара</param>

[ExcludeFromCodeCoverage]

public static void FindElementByValue(MyCollection collection, int amount)

{

FindElementRecursion(collection.Root, amount);

}

/// <summary>

/// Всопмогательная рекурсивная функция для поиска элемента по индексу

/// </summary>

/// <param name="root">Корень дерева</param>

/// <param name="index">Индекс</param>

[ExcludeFromCodeCoverage]

public static void FindElementByIndexRecursion(Point root, int index)

{

if (root != null)

{

FindElementByIndexRecursion(root.Left, index);

if (root.Index==index)

{

Console.WriteLine(root);

}

FindElementByIndexRecursion(root.Right, index);

}

}

/// <summary>

/// Функция поиска элемента по индексу

/// </summary>

/// <param name="collection">Коллекция, где будет произведён поиск</param>

/// <param name="index">Индекс</param>

[ExcludeFromCodeCoverage]

public static void FindElementByIndex(MyCollection collection, int index)

{

FindElementByIndexRecursion(collection.Root, index);

}

/// <summary>

/// Функция поверхностного копирования коллекции

/// </summary>

/// <returns>Возвращает копию коллекции в object</returns>

public object ShallowCopy()

{

return this.MemberwiseClone();

}

/// <summary>

/// Вспомогательная рекурсивная функция для индексации коллекции

/// </summary>

/// <param name="root">Корень дерева</param>

/// <param name="index">Индекс элемента</param>

static void MakeIndexRecursion(Point root, ref int index)

{

if (root!=null)

{

MakeIndexRecursion(root.Left, ref index);

index++;

root.Index = index;

MakeIndexRecursion(root.Right, ref index);

}

}

/// <summary>

/// Функция для индексации коллекции

/// </summary>

/// <param name="collection">Коллекция, в которой будут поставлены индексы</param>

static void MakeIndex(MyCollection collection)

{

if (collection.Root!=null)

{

int index = 0;

MakeIndexRecursion(collection.Root, ref index);

}

}

/// <summary>

/// Функция создания списка элементов коллекции

/// </summary>

/// <param name="root">корень коллекции</param>

/// <returns></returns>

[ExcludeFromCodeCoverage]

List<Point> MakeListValues(Point root)

{

List<Point> list = CreateNotSortedList(root);

return list;

}

/// <summary>

/// Вспомогательная функция, необходимая для создания списка элементов коллекции

/// </summary>

/// <param name="root">Корень дерева</param>

/// <returns>Возвращает список с элементами</returns>

[ExcludeFromCodeCoverage]

public static List<Point> CreateNotSortedList(Point root)

{

List<Point> list = new List<Point>();

if (root != null)

{

AddElementToList(root, ref list);

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

{

list[i].Index = i+1;

list[i].Left = null;

list[i].Right = null;

}

return list;

}

else

{

return null;

}

}

/// <summary>

/// Функция нахождения нумератора

/// </summary>

/// <returns>Возвращает Нумератор</returns>

[ExcludeFromCodeCoverage]

public IEnumerator GetEnumerator()

{

int i = 0;

while (i < Count)

{

yield return Values[i];

i++;

}

}

/// <summary>

/// Функция глубокого клонирования коллекции

/// </summary>

/// <returns>Возвращает склонированную коллекцию в типе object</returns>

public object Clone()

{

if (this.Root!=null)

{

MyCollection tempCollection = new MyCollection(this.Size);

tempCollection.Root = this.Root;

tempCollection.Size = this.Size;

List<Point> list = MyCollection.CreateList(this.Root);

Point root = this.Root;

MyCollection.SortTree(ref root, ref list);

this.Root = root;

MyCollection.MakeIndex(this);

return tempCollection;

}

else

{

return null;

}

}

public void CopyTo(Array array, int index) // Функция, необходимая для реализации интерфейса ICollection

{

throw new NotImplementedException();

}

/// <summary>

/// Перегрузка функции сравнения для UNIT-тестов

/// </summary>

/// <param name="obj">Объект сравнения</param>

/// <returns>Результат в true или false</returns>

public override bool Equals(object? obj)

{

if (obj is MyCollection myCollection)

{

return ((this.Size == myCollection.Size));

}

else

{

return false;

}

}

}

}

# ПРИЛОЖЕНИЕ Р **Программа MenuActivity.cs (часть 4)**

using Microsoft.VisualBasic;

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Diagnostics.CodeAnalysis;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab12\_part4

{

/// <summary>

/// Класс функций основного меню

/// </summary>

[ExcludeFromCodeCoverage]

internal class MenuActivity

{

/// <summary>

/// Функция создания пустой коллекции с заданным размером

/// </summary>

/// <returns>Пустая коллекция с размером capacity</returns>

public static MyCollection CreateEmptyCollection()

{

Console.WriteLine(@"[1] - Создать MyCollection()

[2] - Создать MyCollection(capacity)");

int input = AdditionalActions.InputNumberMenu();

MyCollection collection;

Console.Clear();

switch (input)

{

case 1: // Пустая без capacity

collection= new MyCollection();

AdditionalActions.ContinueProgram();

return collection;

break;

case 2: // Пустая с capacity

Console.WriteLine("Введите целое число - размер коллекции:");

int capacity = AdditionalActions.InputPositiveNumber();

collection = new MyCollection(capacity);

AdditionalActions.ContinueProgram();

return collection;

break;

default:

return null;

break;

}

}

/// <summary>

/// Функция создания коллекции путём использования другой

/// </summary>

/// <returns>Новая коллекция с элементами старой</returns>

public static MyCollection CreateCollection()

{

Console.WriteLine("Введите целое число - размер коллекции, на основе которой будет создана новая:");

int capacity = AdditionalActions.InputPositiveNumber();

MyCollection collection = new MyCollection(capacity);

MyCollection.AddFewElementsToCollection(collection, capacity);

AdditionalActions.ContinueProgram();

return collection;

}

/// <summary>

/// Добавление элементов с заданным количеством

/// </summary>

/// <param name="collection">Коллеция, куда будут добавляться элементы</param>

/// <returns>Коллекция с новым элементами</returns>

public static MyCollection AddElements(MyCollection collection)

{

Console.WriteLine("Введите целое число - количество элементов, которое будет добавлено");

int elements = AdditionalActions.InputPositiveNumber();

MyCollection.AddFewElementsToCollection(collection, elements);

AdditionalActions.ContinueProgram();

return collection;

}

/// <summary>

/// Удаление элементов из коллекции по первой букве

/// </summary>

/// <param name="collection">Коллекция, в которой будут удалены элементы</param>

/// <returns>коллекцию без удалённых элементов</returns>

public static MyCollection RemoveElements(ref MyCollection collection)

{

MyCollection.RemoveElementFromCollectionBySymbol(ref collection);

Console.Clear();

AdditionalActions.ContinueProgram();

return collection;

}

/// <summary>

///

/// </summary>

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

public static void FindByValue(MyCollection collection)

{

Console.WriteLine("Введите количество товара, по которому будет идти поиск:");

int amount = AdditionalActions.InputPositiveNumber();

MyCollection.FindElementByValue(collection, amount);

AdditionalActions.ContinueProgram();

}

/// <summary>

/// Печать коллекции foreach

/// </summary>

/// <param name="collection">коллекция, которую потребруется распечатать</param>

public static void PrintCollection(MyCollection collection)

{

foreach (Point item in collection)

{

Console.WriteLine(item);

}

AdditionalActions.ContinueProgram();

}

/// <summary>

/// Глубокое клонирование

/// </summary>

/// <param name="collection">коллекция, которая будет склонирована</param>

public static void Clone(MyCollection collection)

{

MyCollection tempCollection = (MyCollection)collection.Clone();

Console.WriteLine("Введите новое имя для корня для проверки клонирования:");

string name = Console.ReadLine();

collection.Root.Info.Name = name;

Console.WriteLine(tempCollection.Root);

AdditionalActions.ContinueProgram();

}

/// <summary>

/// Поверхностное копирование

/// </summary>

/// <param name="collection">Коллекция, которую будут копировать</param>

public static void ShallowCopy(MyCollection collection)

{

MyCollection tempCollection = (MyCollection)collection.ShallowCopy();

Console.WriteLine("Введите новое имя для корня для проверки клонирования:");

string name = Console.ReadLine();

collection.Root.Info.Name = name;

Console.WriteLine(tempCollection.Root);

AdditionalActions.ContinueProgram();

}

/// <summary>

/// Нахождение элемента по индексу

/// </summary>

/// <param name="collection">коллекция, в которой будет произведён поиск</param>

public static void FindByIndex(MyCollection collection)

{

Console.WriteLine("Введите индекс, по которому будет идти поиск:");

int index = AdditionalActions.InputPositiveNumber();

MyCollection.FindElementByIndex(collection, index);

AdditionalActions.ContinueProgram();

}

}

}

# ПРИЛОЖЕНИЕ С **Программа AdditionalActions.cs (часть 4)**

using System;

using System.Collections.Generic;

using System.Diagnostics.CodeAnalysis;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab12\_part4

{

/// <summary>

/// Вспомогательный класс для реализации дополнительных функций

/// </summary>

[ExcludeFromCodeCoverage]

internal class AdditionalActions

{

/// <summary>

/// Функция ожидания нажатия клавиши "Enter" для продолжения программы

/// </summary>

public static void ContinueProgram()

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Введите enter, чтобы продолжить.");

ConsoleKey key;

do

{

key =Console.ReadKey().Key;

} while (key != ConsoleKey.Enter);

Console.ForegroundColor = ConsoleColor.White;

Console.Clear();

}

/// <summary>

/// Функция вывода контекстного меню

/// </summary>

public static void PrintMenuText()

{

Console.WriteLine(" - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -");

Console.WriteLine(@$"| Меню: |

| |

| [1] - Создать пустую коллекцию (без размера или заданным размером). |

| [2] - Создать коллекцию на основе другой коллекции. |

| [3] - Добавить элементы. |

| [4] - Удалить элементы по первой букве названия. |

| [5] - Поиск по значению. |

| [6] - Глубокое клонирование. |

| [7] - Поверхностное копирование. |

| [8] - Поиск по индексу. |

| [9] - Печать foreach |

| |

| [0] - Выход. |");

Console.WriteLine(" - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -");

}

/// <summary>

/// Функция ввода целого числа для выбора пунктов меню

/// </summary>

/// <returns>Целое число</returns>

public static int InputNumberMenu()

{

int enter;

bool isInt = false;

do

{

isInt = int.TryParse(Console.ReadLine(), out enter);

} while (isInt != true);

return enter;

}

/// <summary>

///Функция ввода положительного числа

/// </summary>

/// <returns>Целое число большее 0</returns>

public static int InputPositiveNumber()

{

int enter;

bool isInt = false;

bool isOk = false;

do

{

isInt = int.TryParse(Console.ReadLine(), out enter);

isOk = enter>0;

} while ((isInt && isOk) != true);

return enter;

}

}

}

# ПРИЛОЖЕНИЕ Т **Программа UnitTest1.cs (Юнит-Тесты)**

using ClassLibraryLab10;

using lab;

using Lab12\_part3;

using lab12\_part4;

using Microsoft.VisualBasic;

using Microsoft.VisualStudio.TestPlatform.CommunicationUtilities.ObjectModel;

using Newtonsoft.Json.Linq;

using System;

using System.Collections.ObjectModel;

using Point = lab12\_part4.Point;

namespace lab.Tests

{

[TestClass]

public class UnitTest1

{

[TestMethod]

public void TestMyCollection1()

{

MyCollection collection1 = new MyCollection(6);

MyCollection.AddFewElementsToCollection(collection1, 6);

MyCollection collection2 = (MyCollection)collection1.Clone();

Assert.AreEqual(collection1, collection2);

}

[TestMethod]

public void TestMyCollection2()

{

MyCollection collection1 = new MyCollection(6);

MyCollection.AddFewElementsToCollection(collection1, 6);

MyCollection collection2 = (MyCollection)collection1.ShallowCopy();

collection1.Root.Info.Amount = 120;

Assert.AreEqual(collection1.Root.Info.Amount, collection2.Root.Info.Amount);

}

[TestMethod]

public void TestMyCollection3()

{

MyCollection collection1 = new MyCollection(6);

MyCollection.AddFewElementsToCollection(collection1, 6);

List<lab12\_part4.Point> list = new List<Point>();

Point root = collection1.Root;

collection1.Root.Info.Name = "sir";

MyCollection.CountElementsBySymbol(collection1.Root, ref root, "s", ref list);

Assert.AreEqual(list.Count, 5);

}

[TestMethod]

public void TestMyCollection4()

{

MyCollection collection1 = new MyCollection(6);

MyCollection.AddFewElementsToCollection(collection1, 3);

MyCollection.CountElements(collection1);

Assert.AreEqual(3, collection1.Count);

}

[TestMethod]

public void TestMyCollection5()

{

MyCollection collection1 = new MyCollection(6);

MyCollection.AddFewElementsToCollection(collection1, 6);

MyCollection collection2 = new MyCollection(collection1);

Assert.AreEqual(collection2, collection1);

}

[TestMethod]

public void TestMyCollection6()

{

MyCollection collection = new MyCollection();

Assert.AreEqual(0, collection.Size);

}

[TestMethod]

public void TestHashTable1()

{

Toy toy = new Toy();

toy.RandomInit();

Lab12\_part3.Point point = new Lab12\_part3.Point(toy);

point = Lab12\_part3.Point.Delete();

Assert.IsNull(point);

}

[TestMethod]

public void TestHashTable2()

{

HTable hTable = new HTable(5);

Assert.AreEqual(5, hTable.Size);

}

[TestMethod]

public void TestHashTable3()

{

HTable hTable = new HTable(0);

Toy toy = new Toy();

toy.RandomInit();

bool ok = hTable.AddElement(ref hTable, toy, 0);

Assert.IsTrue(ok);

}

[TestMethod]

public void TestHashTable4()

{

Toy toy = new Toy();

toy.RandomInit();

Point [] Table = new Point[2];

Table[0] = new Point(toy, 1);

Table[1] = (Point)Table[0].Clone();

Assert.AreNotSame(Table[0], Table[1]);

}

[TestMethod]

public void TestHashTable5()

{

Toy toy = new Toy(123, 100, "Name", "type");

Lab12\_part3.Point temp = new Lab12\_part3.Point(toy);

int index = temp.GetHashCode();

Assert.AreEqual(6, index);

}

}

}

# ПРИЛОЖЕНИЕ У **Покрытие кода Fine Code Coverage**

