|  |
| --- |
| 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 №14**  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)

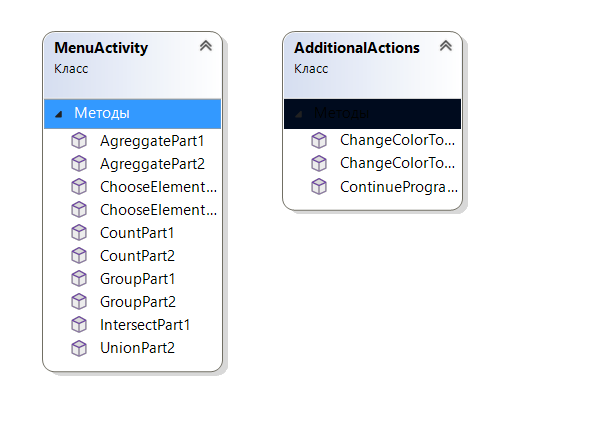
**Часть 1 Создание запросов для обобщённой стандартной коллекции**

**1 Постановка задачи**

1. Сформировать обобщенную стандартную коллекцию, содержащую ссылки на другие стандартные обобщенные коллекции.
2. Заполнить коллекции объектами иерархии классов (лабораторная работа №10).
3. Выполнить запросы функции (всего должно быть выполнено не менее 5 запросов):
   1. На выборку данных.
   2. Получение счетчика (количества объектов с заданным параметром).
   3. Использование операций над множествами (пересечение, объединение, разность).
   4. Агрегирование данных.
   5. Группировка данных
4. Запросы должны быть выполнены двумя способами:
   1. С использованием LINQ запросов.
   2. С использованием методов расширения.
5. Каждый запрос выполняется в отдельной функции.
6. Примеры запросов (лабораторная работа №10).

7 вариант – Коллекция Магазин (Queue)

**2 Диаграмма классов**



**Рисунок 1 – Диаграмма классов**

Программа состоит из двух классов (Рисунок 1), а именно:

* MenuActivity – Этот основной класс хранит в себе все LINQ-запросы и запросы метода расширения.
* AdditionalActions – Вспомогательный класс, который хранит в себе дополнительные необязательные функции.

**3 Код функций для реализации запросов**

Код представлен представлен в классе MenuActivity в приложении. См. Приложение В.

Этот класс содержит пять функций с запросами:

* ChooseElementPart1 – этот запрос выполняет выборку элементов по свойству количества элементов(Amount), где количество элементов больше 500.
* IntersectPart1 – этот запрос выполняет поиск одинаковых элементов в двух очередях.
* CountPart1 – этот запрос считает количество элементов по информационному полю типа (ToyType) «Модель».
* AgreggatePart1 – этот запрос выполняет поиск максимальной цены по информационному полю цены (Price).
* GroupPart1 – запрос выполняет группировку элементов по информационному полю типа (ToyType).

**4 Код программы (листинг)**

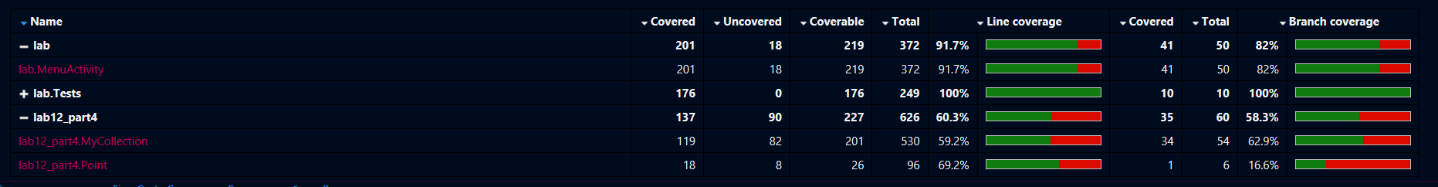
Код программы представлен в приложении. См. Приложение А-В

**5 Код Unit-тестов**

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

**6 Анализ покрытия кода тестами**

С помощью Unit-тестов удалось достичь высокий процент покрытия кода, так как тесты проводились на основе одинаковости ответов в методе расширениях и LINQ-запросов. (Рисунок 2).



**Рисунок 2 – Покрытие кода тестами**

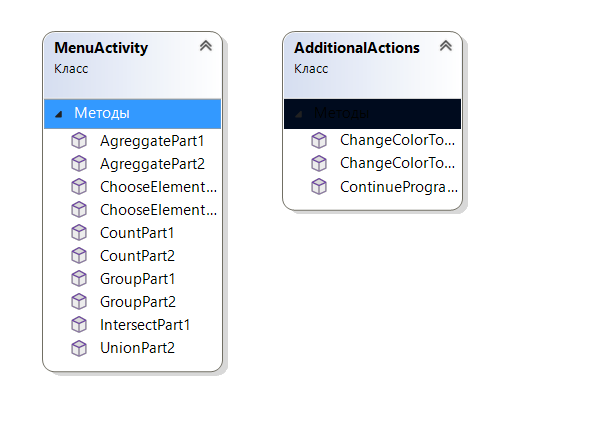
**Часть 2 Создание запросов для коллекции из лабораторной работы №12**

**1 Постановка задачи**

1. К коллекции из лабораторной работы 12 применить запросы:
   1. На выборку данных.
   2. Получение счетчика (количества объектов с заданным параметром).
   3. Использование операций над множествами (пересечение, объединение, разность).
   4. Агрегирование данных.
   5. Группировка данных
2. Запросы должны быть выполнены двумя способами:
   1. С использованием LINQ запросов.
   2. С использованием методов расширения.

7 вариант – Коллекция Отдел (Dictionary).

**2 Диаграмма классов**



**Рисунок 3 – Диаграмма классов**

Программа состоит из двух классов (Рисунок 3), а именно:

* MenuActivity – Этот основной класс хранит в себе все LINQ-запросы и запросы метода расширения.
* AdditionalActions – Вспомогательный класс, который хранит в себе дополнительные необязательные функции.

**3 Код функций для реализации запросов**

Код представлен представлен в классе MenuActivity в приложении. См. Приложение В.

Этот класс содержит пять функций с запросами:

* ChooseElementPart2 – этот запрос выполняет выборку элементов по свойству количества элементов(Amount), где количество элементов меньше или равно 500.
* UnionPart2 – этот запрос выполняет объединение элементов в двух коллекциях.
* CountPart2 – этот запрос считает количество элементов по информационному полю типа (ToyType) «Головоломка».
* AgreggatePart2 – этот запрос выполняет поиск минимальной цены по информационному полю цены (Price).
* GroupPart2 – запрос выполняет группировку элементов по информационному полю типа (ToyType).

**4 Код программы (листинг)**

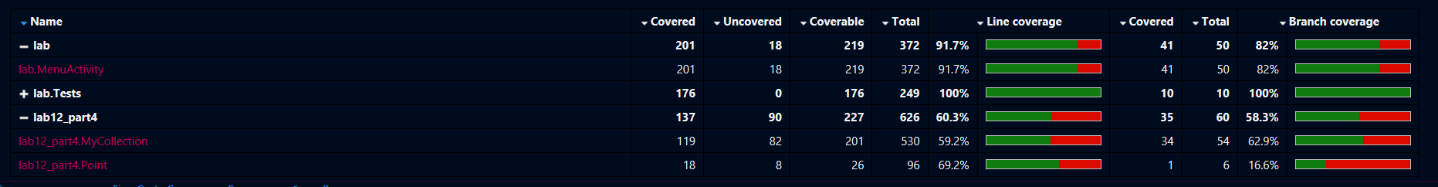
Код программы представлен в приложении. См. Приложение А-В

**5 Код Unit-тестов**

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

**6 Анализ покрытия кода тестами**

С помощью Unit-тестов удалось достичь высокий процент покрытия кода, так как тесты проводились на основе одинаковости ответов в методе расширениях и LINQ-запросов. (Рисунок 4).



**Рисунок 4 – Покрытие кода тестами**

# ПРИЛОЖЕНИЕ А **Программа Program.cs**

using ClassLibraryLab10;

using lab12\_part4;

using System.Diagnostics.CodeAnalysis;

using System.Linq;

using System.Threading.Channels;

namespace lab // Вариант 7

{

internal class Program

{

[ExcludeFromCodeCoverage]

static void Main(string[] args)

{

//Часть 1

Queue<Queue<Toy>> shop = new Queue<Queue<Toy>>();

Queue<Toy> toyQ1 = new Queue<Toy>();

Queue<Toy> toyQ2 = new Queue<Toy>();

Queue<Toy> toyQ3 = new Queue<Toy>();

Toy tempToy = new Toy();

for (int i = 0; i < 10; i++)

{

tempToy = new Toy();

tempToy.RandomInit();

toyQ1.Enqueue(tempToy);

tempToy = new Toy();

tempToy.RandomInit();

toyQ2.Enqueue(tempToy);

tempToy = new Toy();

tempToy.RandomInit();

toyQ3.Enqueue(tempToy);

}

shop.Enqueue(toyQ1);

shop.Enqueue(toyQ2);

shop.Enqueue(toyQ3);

MenuActivity.ChooseElementPart1(shop);

AdditionalActions.ContinueProgram();

MenuActivity.CountPart1(shop);

AdditionalActions.ContinueProgram();

tempToy = new Toy();

tempToy.RandomInit();

toyQ1.Enqueue(tempToy);

toyQ2.Enqueue(tempToy);

MenuActivity.IntersectPart1(toyQ1, toyQ2);

AdditionalActions.ContinueProgram();

MenuActivity.AgreggatePart1(shop);

AdditionalActions.ContinueProgram();

MenuActivity.GroupPart1(shop);

AdditionalActions.ContinueProgram();

//Часть 2

MyCollection myCollection = new MyCollection(10);

MyCollection.AddFewElementsToCollection(myCollection, 10);

var myCollectionEnum = myCollection.OfType<Point>();

MenuActivity.ChooseElementPart2(myCollectionEnum);

AdditionalActions.ContinueProgram();

MenuActivity.CountPart2(myCollectionEnum);

AdditionalActions.ContinueProgram();

MyCollection myCollection2 = new MyCollection(10);

MyCollection.AddFewElementsToCollection(myCollection2, 10);

var myCollectionEnum2 = myCollection2.OfType<Point>();

MenuActivity.UnionPart2(myCollectionEnum, myCollectionEnum2);

AdditionalActions.ContinueProgram();

MenuActivity.AgreggatePart2(myCollectionEnum);

AdditionalActions.ContinueProgram();

MenuActivity.GroupPart2(myCollectionEnum);

AdditionalActions.ContinueProgram();

}

}

}}

# ПРИЛОЖЕНИЕ Б **Программа MenuActivity.cs**

using ClassLibraryLab10;

using lab12\_part4;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab

{

/// <summary>

/// Класс, содержащий все запросы с помощью LINQ и методами расширения

/// </summary>

public class MenuActivity

{

/// <summary>

/// Первый запрос для первой части, делающий выборку элементов с количеством единиц более, чем в 500

/// </summary>

/// <param name="shop">Очередь с элементами</param>

/// <returns>Логическое выражение для проверки</returns>

public static bool ChooseElementPart1(Queue<Queue<Toy>> shop)

{

var resultLinq = from item in shop

from toy in item

where toy.Amount>500

select toy;

var resultExtension = shop.SelectMany(toy => toy)

.Where(toy => toy.Amount>500);

Console.WriteLine("Элементы с количеством большим, чем 500:");

Console.WriteLine();

foreach (Toy item in resultLinq)

{

Console.WriteLine(item);

}

AdditionalActions.ChangeColorToGreen();

foreach (Toy item in resultExtension)

{

Console.WriteLine(item);

}

if (resultLinq.Count() == resultExtension.Count())

{

return true;

}

else

{

return false;

}

}

/// <summary>

/// Первый запрос для второй части, делающий выборку элементов с количеством менее 501

/// </summary>

/// <param name="myCollectionEnum">Коллекция с элементами</param>

/// <returns>Логическое выражение для тестов</returns>

public static bool ChooseElementPart2(IEnumerable<Point> myCollectionEnum)

{

var resultLinqPart2 = from item in myCollectionEnum

where item.Info.Amount<=500

select item;

var resultExtensionPart2 = myCollectionEnum.Select(point => point)

.Where(point => point.Info.Amount<=500);

Console.WriteLine("Элементы с количеством меньшим или равно 500:");

Console.WriteLine();

foreach (Point item in resultLinqPart2)

{

Console.WriteLine(item);

}

AdditionalActions.ChangeColorToGreen();

foreach (Point item in resultExtensionPart2)

{

Console.WriteLine(item);

}

if (resultLinqPart2.Count() == resultExtensionPart2.Count())

{

return true;

}

else

{

return false;

}

}

/// <summary>

/// Третий запрос для первой части, определяющий одиннаковые элементы, находящиеся в двух очередях

/// </summary>

/// <param name="toyQ1">Первая очередь с элементами</param>

/// <param name="toyQ2">Вторая очередь</param>

/// <returns>Логическое выражение для проверки</returns>

public static bool IntersectPart1(Queue<Toy> toyQ1, Queue<Toy> toyQ2)

{

var intersectLinq = (from item1 in toyQ1 select item1).Intersect(from item2 in toyQ2 select item2);

var intersectExtension = toyQ1.Select(toy => toy)

.Intersect(toyQ2

.Select(toy => toy));

Console.WriteLine("Пересекающиеся элементы в очередях:");

Console.WriteLine();

foreach (Toy item in intersectLinq)

{

Console.WriteLine(item);

}

AdditionalActions.ChangeColorToGreen();

foreach (Toy item in intersectExtension)

{

Console.WriteLine(item);

}

if (intersectExtension.Count() == intersectLinq.Count())

{

return true;

}

else

{

return false;

}

}

/// <summary>

/// Третий запрос для второй части, объединяющий элементы двух коллекций

/// </summary>

/// <param name="myCollectionEnum">Первая коллекция, состоящая из элементов</param>

/// <param name="myCollectionEnum2">Вторая коллекция, состоящая из элементов</param>

/// <returns>Логическое выражение для проверки</returns>

public static bool UnionPart2(IEnumerable<Point> myCollectionEnum, IEnumerable<Point> myCollectionEnum2)

{

var intersectLinqPart2 = (from item1 in myCollectionEnum select item1).Union(

from item2 in myCollectionEnum2 select item2);

var intersectExtensionPart2 = myCollectionEnum.Select(item => item)

.Union(myCollectionEnum2

.Select(item => item));

Console.WriteLine("Объединение двух коллекций:");

Console.WriteLine();

foreach (Point item in intersectLinqPart2)

{

Console.WriteLine(item);

}

AdditionalActions.ChangeColorToGreen();

foreach (Point item in intersectExtensionPart2)

{

Console.WriteLine(item);

}

if (intersectExtensionPart2.Count() == intersectLinqPart2.Count())

{

return true;

}

else

{

return false;

}

}

/// <summary>

/// Второй запрос для первой части, считающий количество элементов типа "Модель"

/// </summary>

/// <param name="shop">очередь элементов</param>

/// <returns>Логическое выражение для проверки</returns>

public static bool CountPart1(Queue<Queue<Toy>> shop)

{

var countLinq = (from item in shop

from toy in item

where toy.ToyType == "Модель"

select toy).Count();

var countExtension = shop.SelectMany(toy => toy)

.Where(toy => toy.ToyType == "Модель")

.Count();

Console.WriteLine("Количество элементов типа 'Модель':");

Console.WriteLine();

Console.WriteLine("Linq: " + countLinq);

Console.WriteLine("Метод расширения: " + countExtension);

if (countExtension == countLinq)

{

return true;

}

else

{

return false;

}

}

/// <summary>

/// Второй запрос для второй части, считающий количество элементов типа "Головоломка"

/// </summary>

/// <param name="myCollectionEnum">Коллекция элементов</param>

/// <returns>Логическое выражение для проверки</returns>

public static bool CountPart2(IEnumerable<Point> myCollectionEnum)

{

var countLinqPart2 = (from item in myCollectionEnum

where item.Info.ToyType == "Головоломка"

select item).Count();

var countExtensionPart2 = myCollectionEnum.Select(item => item)

.Where(item => item.Info.ToyType == "Головоломка")

.Count();

Console.WriteLine("Количество элементов типа 'Головоломка':");

Console.WriteLine();

Console.WriteLine("Linq: " + countLinqPart2);

Console.WriteLine("Метод расширения: " + countExtensionPart2);

return countExtensionPart2 == countLinqPart2;

}

/// <summary>

/// Четвёртый запрос для первой части, находящий максимальную цену

/// </summary>

/// <param name="shop">Очередь с элементами</param>

/// <returns>Логическое выражение для проверки</returns>

public static bool AgreggatePart1(Queue<Queue<Toy>> shop)

{

var maxLinq = (from item in shop

from toy in item

select toy.Price).Max();

var maxExtension = shop.SelectMany(toy => toy)

.Select(toy => toy.Price)

.Max();

Console.WriteLine("Максимальная цена товара:");

Console.WriteLine();

Console.WriteLine("Linq: " + maxLinq);

Console.WriteLine("Метод расширения: " + maxExtension);

if (maxExtension == maxLinq)

{

return true;

}

else

{

return false;

}

}

/// <summary>

/// Четвёртый запрос для второй части, считающий минимальную цену элементов

/// </summary>

/// <param name="myCollectionEnum">Коллекция с элементами</param>

/// <returns>Логическое выражение для проверки</returns>

public static bool AgreggatePart2(IEnumerable<Point> myCollectionEnum)

{

var maxLinqPart2 = (from item in myCollectionEnum

select item.Info.Price).Min();

var maxExtensionPart2 = myCollectionEnum.Select(item => item)

.Select(item => item.Info.Price)

.Min();

Console.WriteLine("Минимальная цена товара:");

Console.WriteLine( );

Console.WriteLine("Linq: " + maxLinqPart2);

Console.WriteLine("Метод расширения: " + maxExtensionPart2);

if (maxLinqPart2 == maxExtensionPart2)

{

return true;

}

else

{

return false;

}

}

/// <summary>

/// Пятый запрос для первой части, группирующий элементы по типу

/// </summary>

/// <param name="shop">Очередь с элементами</param>

/// <returns>Логическое выражение для проверки</returns>

public static bool GroupPart1(Queue<Queue<Toy>> shop)

{

var groupLinq = from item in shop

from toy in item

group toy by toy.ToyType;

var groupExtension = shop.SelectMany(toy => toy)

.GroupBy(toy => toy.ToyType);

Console.WriteLine("Сгрупированные элементы:");

foreach (IGrouping<string, Toy> item in groupLinq)

{

Console.WriteLine();

Console.WriteLine(item.Key);

foreach (var element in item)

{

Console.WriteLine(element);

}

}

AdditionalActions.ChangeColorToGreen();

foreach (IGrouping<string, Toy> item in groupExtension)

{

Console.WriteLine();

Console.WriteLine(item.Key);

foreach (var element in item)

{

Console.WriteLine(element);

}

}

if (groupExtension.Count() == groupLinq.Count())

{

return true;

}

else

{

return false;

}

}

/// <summary>

/// Пятый запрос для второй части, группирующий элементы по типу

/// </summary>

/// <param name="myCollectionEnum">Коллекция элементов</param>

/// <returns>Логическое выражение для провекри</returns>

public static bool GroupPart2(IEnumerable<Point> myCollectionEnum)

{

var groupLinqPart2 = from item in myCollectionEnum

group item by item.Info.ToyType;

var groupExtensionPart2 = myCollectionEnum.Select(item => item)

.GroupBy(item => item.Info.ToyType);

Console.WriteLine("Сгрупированные элементы:");

foreach (IGrouping<string, Point> item in groupLinqPart2)

{

Console.WriteLine();

Console.WriteLine(item.Key);

foreach (var element in item)

{

Console.WriteLine(element);

}

}

AdditionalActions.ChangeColorToGreen();

foreach (IGrouping<string, Point> item in groupExtensionPart2)

{

Console.WriteLine();

Console.WriteLine(item.Key);

foreach (var element in item)

{

Console.WriteLine(element);

}

}

if (groupExtensionPart2.Count() == groupLinqPart2.Count())

{

return true;

}

else

{

return false;

}

}

}

}

# ПРИЛОЖЕНИЕ В **Программа AdditionalActions.cs**

using System;

using System.Collections.Generic;

using System.Diagnostics.CodeAnalysis;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab

{

/// <summary>

/// Класс вспомогательных функций

/// </summary>

internal class AdditionalActions

{

/// <summary>

/// Функция смены цвета текста в консоли на зелёный

/// </summary>

[ExcludeFromCodeCoverage]

public static void ChangeColorToGreen()

{

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine();

}

/// <summary>

/// Функция смены цвета в консоли на белый

/// </summary>

[ExcludeFromCodeCoverage]

public static void ChangeColorToWhite()

{

Console.ForegroundColor = ConsoleColor.White;

Console.WriteLine();

}

/// <summary>

/// Функция ожидания ввода "Enter" для продолжения

/// </summary>

[ExcludeFromCodeCoverage]

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();

}

}

}

# ПРИЛОЖЕНИЕ Г **Код Unit-Тестов**

using ClassLibraryLab10;

using lab;

using lab12\_part4;

namespace lab.Tests

{

[TestClass]

public class UnitTest1

{

[TestMethod]

public void TestChooseElement1()

{

Queue<Queue<Toy>> shop = new Queue<Queue<Toy>>();

Queue<Toy> toyQ1 = new Queue<Toy>();

Queue<Toy> toyQ2 = new Queue<Toy>();

Queue<Toy> toyQ3 = new Queue<Toy>();

Toy tempToy = new Toy();

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

{

tempToy = new Toy();

tempToy.RandomInit();

toyQ1.Enqueue(tempToy);

tempToy = new Toy();

tempToy.RandomInit();

toyQ2.Enqueue(tempToy);

tempToy = new Toy();

tempToy.RandomInit();

toyQ3.Enqueue(tempToy);

}

shop.Enqueue(toyQ1);

shop.Enqueue(toyQ2);

shop.Enqueue(toyQ3);

bool ok = MenuActivity.ChooseElementPart1(shop);

Assert.IsTrue(ok);

}

[TestMethod]

public void TestChooseElement2()

{

MyCollection myCollection = new MyCollection(10);

MyCollection.AddFewElementsToCollection(myCollection, 10);

var myCollectionEnum = myCollection.OfType<Point>();

bool ok = MenuActivity.ChooseElementPart2(myCollectionEnum);

Assert.IsTrue(ok);

}

[TestMethod]

public void TestCount1()

{

Queue<Queue<Toy>> shop = new Queue<Queue<Toy>>();

Queue<Toy> toyQ1 = new Queue<Toy>();

Queue<Toy> toyQ2 = new Queue<Toy>();

Queue<Toy> toyQ3 = new Queue<Toy>();

Toy tempToy = new Toy();

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

{

tempToy = new Toy();

tempToy.RandomInit();

toyQ1.Enqueue(tempToy);

tempToy = new Toy();

tempToy.RandomInit();

toyQ2.Enqueue(tempToy);

tempToy = new Toy();

tempToy.RandomInit();

toyQ3.Enqueue(tempToy);

}

shop.Enqueue(toyQ1);

shop.Enqueue(toyQ2);

shop.Enqueue(toyQ3);

bool ok = MenuActivity.CountPart1(shop);

Assert.IsTrue(ok);

}

[TestMethod]

public void TestCount2()

{

MyCollection myCollection = new MyCollection(10);

MyCollection.AddFewElementsToCollection(myCollection, 10);

var myCollectionEnum = myCollection.OfType<Point>();

bool ok = MenuActivity.CountPart2(myCollectionEnum);

Assert.IsTrue(ok);

}

[TestMethod]

public void TestIntersect1()

{

Queue<Queue<Toy>> shop = new Queue<Queue<Toy>>();

Queue<Toy> toyQ1 = new Queue<Toy>();

Queue<Toy> toyQ2 = new Queue<Toy>();

Queue<Toy> toyQ3 = new Queue<Toy>();

Toy tempToy = new Toy();

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

{

tempToy = new Toy();

tempToy.RandomInit();

toyQ1.Enqueue(tempToy);

tempToy = new Toy();

tempToy.RandomInit();

toyQ2.Enqueue(tempToy);

tempToy = new Toy();

tempToy.RandomInit();

toyQ3.Enqueue(tempToy);

}

shop.Enqueue(toyQ1);

shop.Enqueue(toyQ2);

shop.Enqueue(toyQ3);

tempToy = new Toy();

tempToy.RandomInit();

toyQ1.Enqueue(tempToy);

toyQ2.Enqueue(tempToy);

bool ok = MenuActivity.IntersectPart1(toyQ1, toyQ2);

Assert.IsTrue(ok);

}

[TestMethod]

public void TestUnion2()

{

MyCollection myCollection = new MyCollection(10);

MyCollection.AddFewElementsToCollection(myCollection, 10);

var myCollectionEnum = myCollection.OfType<Point>();

MyCollection myCollection2 = new MyCollection(10);

MyCollection.AddFewElementsToCollection(myCollection2, 10);

var myCollectionEnum2 = myCollection2.OfType<Point>();

bool ok = MenuActivity.UnionPart2(myCollectionEnum, myCollectionEnum2);

Assert.IsTrue(ok);

}

[TestMethod]

public void TestAgreggate1()

{

Queue<Queue<Toy>> shop = new Queue<Queue<Toy>>();

Queue<Toy> toyQ1 = new Queue<Toy>();

Queue<Toy> toyQ2 = new Queue<Toy>();

Queue<Toy> toyQ3 = new Queue<Toy>();

Toy tempToy = new Toy();

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

{

tempToy = new Toy();

tempToy.RandomInit();

toyQ1.Enqueue(tempToy);

tempToy = new Toy();

tempToy.RandomInit();

toyQ2.Enqueue(tempToy);

tempToy = new Toy();

tempToy.RandomInit();

toyQ3.Enqueue(tempToy);

}

shop.Enqueue(toyQ1);

shop.Enqueue(toyQ2);

shop.Enqueue(toyQ3);

tempToy = new Toy();

tempToy.RandomInit();

toyQ1.Enqueue(tempToy);

toyQ2.Enqueue(tempToy);

bool ok = MenuActivity.AgreggatePart1(shop);

Assert.IsTrue(ok);

}

[TestMethod]

public void TestAgreggate2()

{

MyCollection myCollection = new MyCollection(10);

MyCollection.AddFewElementsToCollection(myCollection, 10);

var myCollectionEnum = myCollection.OfType<Point>();

MyCollection myCollection2 = new MyCollection(10);

MyCollection.AddFewElementsToCollection(myCollection2, 10);

var myCollectionEnum2 = myCollection2.OfType<Point>();

bool ok = MenuActivity.AgreggatePart2(myCollectionEnum);

Assert.IsTrue(ok);

}

[TestMethod]

public void TestGroupBy1()

{

Queue<Queue<Toy>> shop = new Queue<Queue<Toy>>();

Queue<Toy> toyQ1 = new Queue<Toy>();

Queue<Toy> toyQ2 = new Queue<Toy>();

Queue<Toy> toyQ3 = new Queue<Toy>();

Toy tempToy = new Toy();

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

{

tempToy = new Toy();

tempToy.RandomInit();

toyQ1.Enqueue(tempToy);

tempToy = new Toy();

tempToy.RandomInit();

toyQ2.Enqueue(tempToy);

tempToy = new Toy();

tempToy.RandomInit();

toyQ3.Enqueue(tempToy);

}

shop.Enqueue(toyQ1);

shop.Enqueue(toyQ2);

shop.Enqueue(toyQ3);

tempToy = new Toy();

tempToy.RandomInit();

toyQ1.Enqueue(tempToy);

toyQ2.Enqueue(tempToy);

bool ok = MenuActivity.GroupPart1(shop);

Assert.IsTrue(ok);

}

[TestMethod]

public void TestGroupBy2()

{

MyCollection myCollection = new MyCollection(10);

MyCollection.AddFewElementsToCollection(myCollection, 10);

var myCollectionEnum = myCollection.OfType<Point>();

MyCollection myCollection2 = new MyCollection(10);

MyCollection.AddFewElementsToCollection(myCollection2, 10);

var myCollectionEnum2 = myCollection2.OfType<Point>();

bool ok = MenuActivity.GroupPart2(myCollectionEnum);

Assert.IsTrue(ok);

}

}

}