|  |
| --- |
| 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 №11**  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, 2 Работа с коллекцией и обобщённой коллекцией** 3](#_Toc132134439)

[**1 Постановка задачи** 3](#_Toc132134440)

[**2 Диаграмма классов** 4](#_Toc132134441)

[**3 Описание используемых методов** 5](#_Toc132134442)

[**4 Программа (листинг)** 6](#_Toc132134443)

[**5 Код Unit-тестов** 7](#_Toc132134444)

[**6 Анализ покрытия кода тестами** 9](#_Toc132134445)

[**Часть №3 Создание четырёх разных одинаковых коллекций и определение времени поиска элементов.** 10](#_Toc132134446)

[**1 Постановка задачи** 10](#_Toc132134447)

[**2 Диаграмма классов** 12](#_Toc132134448)

[**3 Программа (листинг)** 13](#_Toc132134449)

[**4 Пример демонстрационной программы** 14](#_Toc132134450)

[**5 Код Unit-тестов** 15](#_Toc132134451)

[**6 Анализ покрытия кода тестами** 17](#_Toc132134452)

[ПРИЛОЖЕНИЕ А **Программа Program.cs (часть 1,2)** 18](#_Toc132134453)

[ПРИЛОЖЕНИЕ Б **Программа Program.cs(Часть 3)** 29](#_Toc132134454)

[ПРИЛОЖЕНИЕ В **Программа Goods.cs** 30](#_Toc132134455)

[ПРИЛОЖЕНИЕ Г **Программа Product.cs** 33](#_Toc132134456)

[ПРИЛОЖЕНИЕ Д **Программа MilkProduct.cs** 35](#_Toc132134457)

[ПРИЛОЖЕНИЕ Е **Программа Toy.cs** 38](#_Toc132134458)

[ПРИЛОЖЕНИЕ Ё **Программа Animal.cs** 41](#_Toc132134459)

[ПРИЛОЖЕНИЕ Ж **Программа SortByPrice.cs** 43](#_Toc132134460)

[ПРИЛОЖЕНИЕ З **Программа IInit.cs** 44](#_Toc132134461)

[ПРИЛОЖЕНИЕ И **Программа TestCollections.cs** 45](#_Toc132134462)

**Часть №1, 2 Работа с коллекцией и обобщённой коллекцией**

**1 Постановка задачи**

Часть 1:

1. Создать коллекцию, в которую добавить объекты созданной иерархии классов.
2. Используя меню, реализовать в программе добавление и удаление объектов коллекции.
3. Разработать и реализовать три запроса (количество элементов определенного вида, печать элементов определенного вида и т.п.).
4. Выполнить перебор элементов коллекции с помощью метода foreach.
5. Выполнить клонирование коллекции.
6. Выполнить сортировку коллекции (если коллекция не отсортирована) и поиск заданного элемента в коллекции.

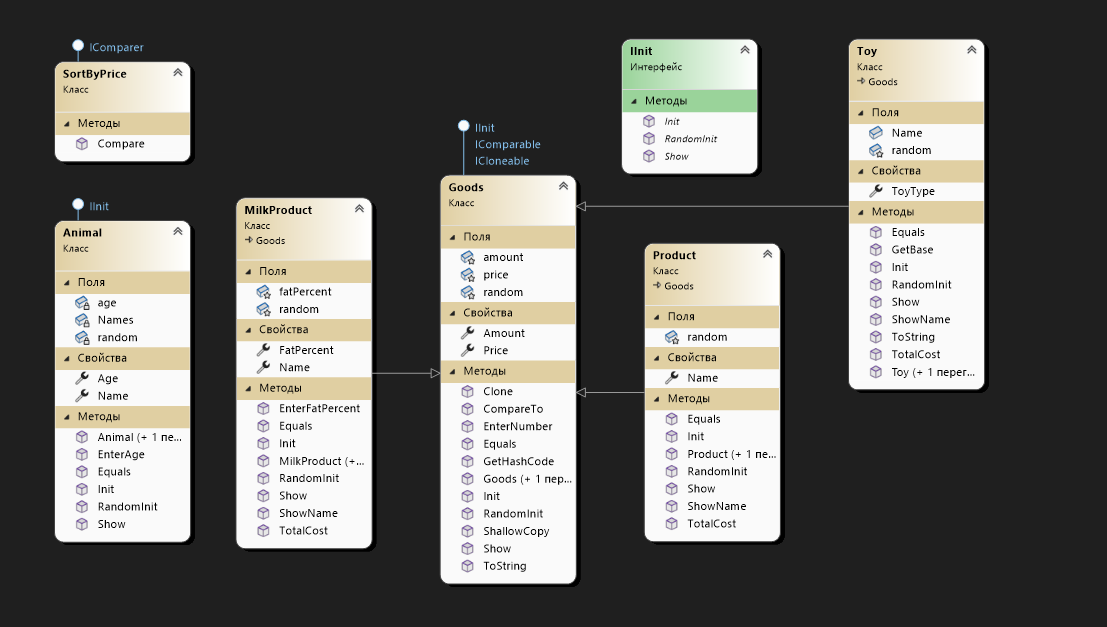
7 вариант – Stack, необходимо создать стандартную коллекцию типа Stack и заполнить её элементами иерархии классов из прошлой работы, а затем выполнить необходимые действия с помощью определённых для Stack методов.

Часть 2:

1. Создать обобщенную коллекцию, в которую добавить объекты созданной иерархии классов.
2. Используя меню, реализовать в программе добавление и удаление объектов коллекции.
3. Разработать и реализовать три запроса (количество элементов определенного вида, печать элементов определенного вида и т.п.).
4. Выполнить перебор элементов коллекции с помощью метода foreach.
5. Выполнить клонирование коллекции.
6. Выполнить сортировку коллекции (если коллекция не отсортирована) и поиск заданного элемента в коллекции.

7 вариант – List<T>, необходимо создать обобщённую коллекцию типа List<T> и заполнить её элементами иерархии классов из прошлой работы, а затем выполнить необходимые действия с помощью определённых для List<T> методов.

**2 Диаграмма классов**



**Рисунок 1 – Диаграмма классов**

**3 Описание используемых методов**

Stack.Pop(item) – Удаление и возвращение последнего объекта стека.

Stack.Push(item) – Добавление последнего объекта в стек.

Stack.Count – Подсчёт количества элементов в стеке.

Stack.CopyTo(array, index) – Копирует стек в одномерный массив, начиная с указанного индекса массива.

SortStack(stack) – Добавленная функция, которая сортирует стек по количеству товара(Amount).

List<T>.Count – Подсчёт количества элементов в списке.

List<T>.RemoveAt(index) – Удаление элемента в определённой позиции.

List<T>.Insert(index , item) – Добавление элемента в указанную позицию.

List<T>.BinarySearch(item) – Нахождение элемента в списке бинарным поиском.

List<T>.CopyTo(array , index) – Копирует список в одномерный массив, начиная с указанного индекса массива.

**4 Программа (листинг)**

Код программы представлен в приложении. См. Приложение А

**5 Код Unit-тестов**

using lab;

namespace lab.Tests

{

[TestClass]

public class UnitTest1

{

[TestMethod]

public void TestGoods()

{

Goods expected = new Goods(0, 21);

Goods actual = new Goods(0, 21);

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void TestGoods2()

{

Goods expected = new Goods();

Goods actual = new Goods(0, 0);

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void TestMilkProduct()

{

MilkProduct expected = new MilkProduct(20, 30, "Test", 20);

MilkProduct actual = new MilkProduct(20, 30, "Test", 20);

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void TestMilkProduct2()

{

MilkProduct expected = new MilkProduct();

MilkProduct actual = new MilkProduct(0, 0, "NoName", 0);

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void TestMilkProductTotalCost()

{

MilkProduct expected = new MilkProduct(25, 30, "name", 20);

Assert.AreEqual(expected.TotalCost(), expected.Amount\*expected.Price);

}

[TestMethod]

public void TestProduct()

{

Product expected = new Product(20, 30, "Test");

Product actual = new Product(20, 30, "Test");

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void TestProduct2()

{

Product expected = new Product();

Product actual = new Product(0, 0, "NoName");

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void TestProductTotalCost()

{

Product expected = new Product(25, 30, "name");

Assert.AreEqual(expected.TotalCost(), expected.Amount\*expected.Price);

}

[TestMethod]

public void TestToy()

{

Toy expected = new Toy(20, 30, "Test", "TypeTest");

Toy actual = new Toy(20, 30, "Test", "TypeTest");

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void TestToy2()

{

Toy expected = new Toy();

Toy actual = new Toy(0, 0, "NoName", "NoType");

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void TestToyTotalCost()

{

Toy expected = new Toy(25, 30, "name", "type");

Assert.AreEqual(expected.TotalCost(), expected.Amount\*expected.Price);

}

[TestMethod]

public void TestGoodsClone()

{

Goods expected = new Goods(25, 30);

Goods actual = new Goods(123, 456);

actual= (Goods)expected.Clone();

Assert.AreEqual(expected, actual);

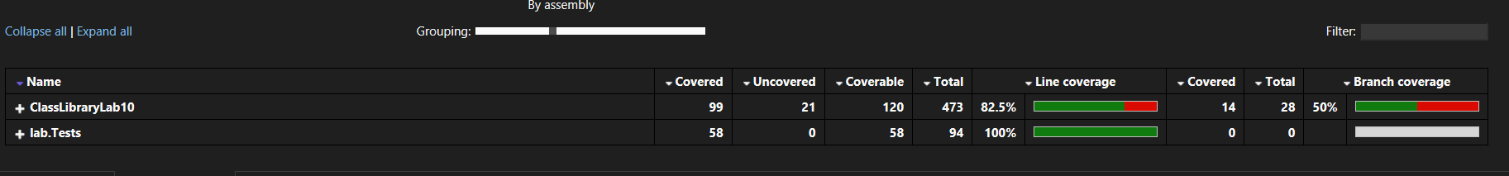
}

}

}

**6 Анализ покрытия кода тестами**

С помощью Unit-тестов удалось достичь покрытия кода в 82%, были сохранены тесты из прошлой работы, так как в этой работе все методы, которые используются для решения, уже определены внутри коллекций, значит, что можно исключить их тестирование, ведь нет необходимости перегружать новые функции.

****

**Рисунок 2 – покрытие кода тестами**

**Часть №3 Создание четырёх разных одинаковых коллекций и определение времени поиска элементов.**

**1 Постановка задачи**

1. Создать иерархию классов (базовый – производный) в соответствии с вариантом (см. лаб. раб. №10).

2. В производном классе определить свойство или метод, которое возвращает ссылку на объект базового класса (это свойство должно возвращать объект базового класса). Например, для иерархии классов Person-Student в классе производном классе Student можно определить свойство

public Person BasePerson

{

get

{

return new Person(name, age);//возвращает объект базового класса

}

}

3. Определить класс TestCollections, который содержит поля следующих типов

Коллекция\_1< TValue > ;

Коллекция\_1<string> ;

Коллекция\_2<TKey, TValue> ;

Коллекция\_2<string, TValue> .

где тип ключа TKey и тип значения TValue связаны отношением базовый-производный (см. задание 1), Коллекция\_1 и Коллекция\_2 – коллекции из пространства имен System.Collections.Generic.

Например:

List<Student> col1=new List<Student>();

List<string> col2=new List<Student>();

Dictionary<Person, Student> col2=new Dictionary<Person, Student>();

Dictionary<string, Student> col2=new Dictionary<string, Student>();

4. Написать конструктор класса TestCollections, в котором создаются коллекции с заданным числом элементов (1000 элементов).

5. Предусмотреть автоматическую генерацию элементов коллекции таким образом, что каждый объект (Student) содержит подобъект базового класса (Person). Все четыре коллекции должны содержать одинаковое число элементов. Сами элементы также должны быть одинаковые, т.е. сгенерировали элемент производного класса и на его основе заполнили элементами все 4 коллекции. В коллекциях, где должны быть строки, для заполнения используется метод ToString().

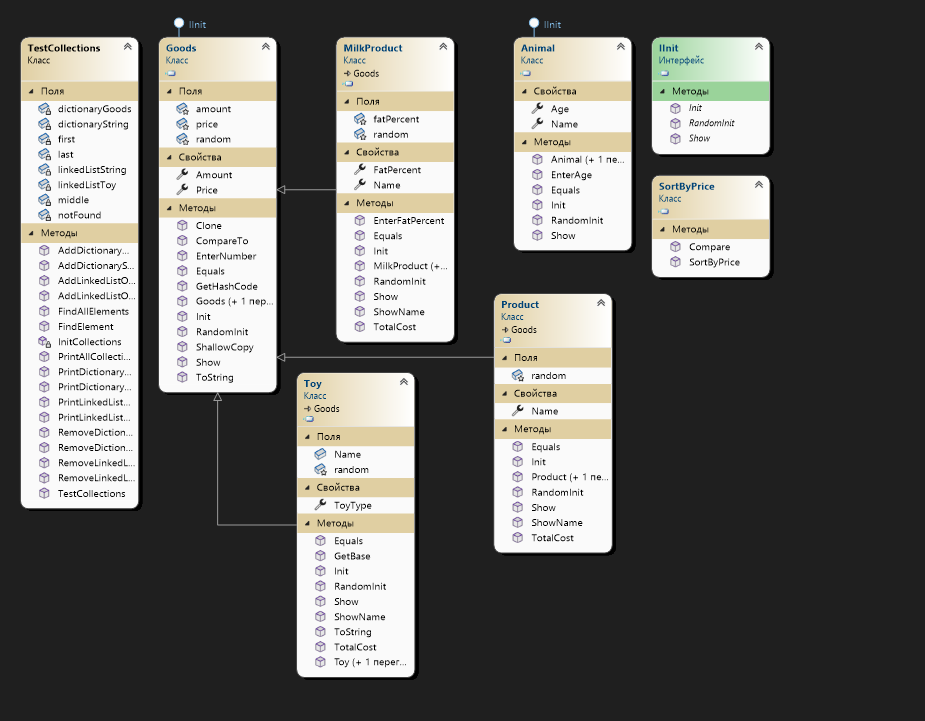
6. Предусмотреть методы для работы с коллекциями (добавление и удаление элементов).

7. Для четырех разных элементов – первого, центрального, последнего и элемента, не входящего в коллекцию – надо измерить время поиска элемента в коллекциях Коллекция\_1< TValue > и Коллекция\_1<string> с помощью метода Contains; элемента по ключу в коллекциях Коллекция\_2< TKey, TValue> и Коллекция\_2 <string, TValue > с помощью метода ContainsKey; значения элемента в коллекции Коллекция\_2< TKey, TValue > с помощью метода ContainsValue. Обратите внимание на то, что искать нужно сами элементы, а не ссылки на них!

8. Объяснить полученные результаты.

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.

**2 Диаграмма классов**



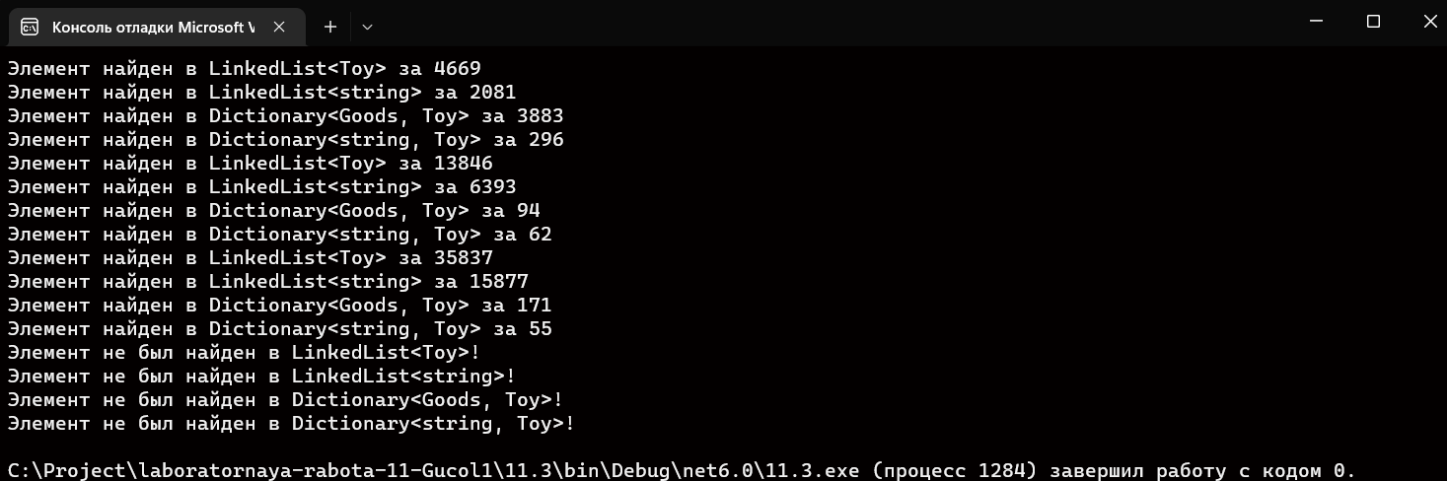
**Рисунок 3 – диаграмма классов**

**3 Программа (листинг)**

Код программы представлен в приложении. См. Приложение Б

**4 Пример демонстрационной программы**

Демонстрационная программа вывела значения при размере коллекций в 10000 объектов.



**Рисунок 4 – Пример демонстрационной работы**

Таким образом, самой быстродействующей коллекцией оказалась Dictionary<string , Toy>, так как поиск элемента выполняется с помощью хэшкода, который очень хорошо оптимизирован для поиска и минимализации затраченного времени.

**5 Код Unit-тестов**

using lab;

namespace lab.Tests

{

[TestClass]

public class UnitTest1

{

[TestMethod]

public void TestGoods()

{

Goods expected = new Goods(0, 21);

Goods actual = new Goods(0, 21);

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void TestGoods2()

{

Goods expected = new Goods();

Goods actual = new Goods(0, 0);

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void TestMilkProduct()

{

MilkProduct expected = new MilkProduct(20, 30, "Test", 20);

MilkProduct actual = new MilkProduct(20, 30, "Test", 20);

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void TestMilkProduct2()

{

MilkProduct expected = new MilkProduct();

MilkProduct actual = new MilkProduct(0, 0, "NoName", 0);

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void TestMilkProductTotalCost()

{

MilkProduct expected = new MilkProduct(25, 30, "name", 20);

Assert.AreEqual(expected.TotalCost(), expected.Amount\*expected.Price);

}

[TestMethod]

public void TestProduct()

{

Product expected = new Product(20, 30, "Test");

Product actual = new Product(20, 30, "Test");

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void TestProduct2()

{

Product expected = new Product();

Product actual = new Product(0, 0, "NoName");

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void TestProductTotalCost()

{

Product expected = new Product(25, 30, "name");

Assert.AreEqual(expected.TotalCost(), expected.Amount\*expected.Price);

}

[TestMethod]

public void TestToy()

{

Toy expected = new Toy(20, 30, "Test", "TypeTest");

Toy actual = new Toy(20, 30, "Test", "TypeTest");

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void TestToy2()

{

Toy expected = new Toy();

Toy actual = new Toy(0, 0, "NoName", "NoType");

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void TestToyTotalCost()

{

Toy expected = new Toy(25, 30, "name", "type");

Assert.AreEqual(expected.TotalCost(), expected.Amount\*expected.Price);

}

[TestMethod]

public void TestGoodsClone()

{

Goods expected = new Goods(25, 30);

Goods actual = new Goods(123, 456);

actual= (Goods)expected.Clone();

Assert.AreEqual(expected, actual);

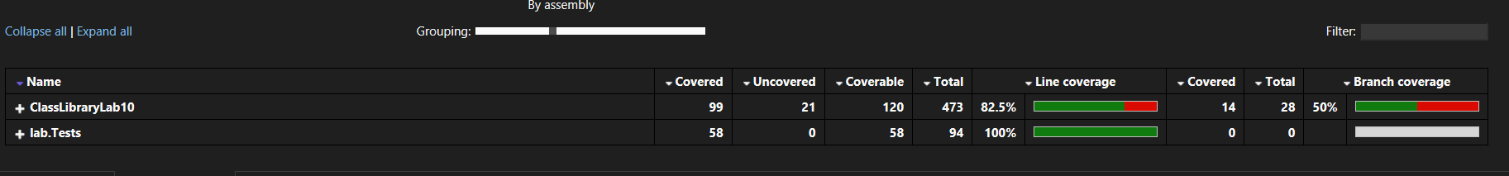
}

}

}

**6 Анализ покрытия кода тестами**

С помощью Unit-тестов удалось достичь покрытия кода в 82%, были сохранены тесты из прошлой работы, так как в этой работе все методы, которые используются для решения, уже определены внутри коллекций, значит, что можно исключить их тестирование, ведь нет необходимости перегружать новые функции.

****

**Рисунок 5 – покрытие кода тестами**

# ПРИЛОЖЕНИЕ А **Программа Program.cs (часть 1,2)**

using ClassLibraryLab10;

using System.Collections;

using System.Collections.Generic;

using System.Diagnostics.CodeAnalysis;

using System.Net;

using System.Transactions;

namespace lab

{

internal class Program

{

// 7 вариант

[ExcludeFromCodeCoverage]

static void Main(string[] args)

{

// 1 Часть

//Инициализация массива товаров из различных категорий

Goods goods1 = new Goods();

goods1.RandomInit();

Goods goods2 = new Goods();

goods2.RandomInit();

Toy toy1 = new Toy();

Toy toy2 = new Toy();

MilkProduct milkProduct1 = new MilkProduct();

MilkProduct milkProduct2 = new MilkProduct();

toy1.RandomInit();

toy2.RandomInit();

milkProduct1.RandomInit();

milkProduct2.RandomInit();

Goods[] goodsArray = { goods1, goods2, toy1, toy2, milkProduct1, milkProduct2 };

//Перенос объектов массива типа Goods в Stack типа Goods

Stack<Goods> stack = new Stack<Goods>(goodsArray);

//Логичесая переменая продолжения алгоритма

bool isNext = false;

int input; //Переменная ввода

do

{

//Вывод стэка

Console.WriteLine("Стэк:");

if (stack.Count == 0)

{

Console.WriteLine("Пусто");

}

foreach (var item in stack)

{

item.Show();

Console.WriteLine("");

}

Console.WriteLine($@"

Меню для Stack

[1] - Удалить объект коллекции

[2] - Добавить случайный объект коллекции

[0] - Дальше");

input = InputNumberMenu();

switch (input)

{

//Удаление объекта стэка

case 1:

if (stack.Count>0)

{

stack.Pop();

}

break;

//Добавление элемента в стэк

case 2:

Console.Clear();

bool isNextAdd = false;

do

{

Console.WriteLine(@"Какого класса добавить объект?

[1] - Product

[2] - Goods

[3] - MilkProduct

[4] - Toy");

input = InputNumberMenu();

switch (input)

{

//Добавление элемента типа Product

case 1:

//AddRandom<Product>();

//void AddRandom<T>(arg1, atg2) where T : Product

//{

//

//}

Product tempProduct = new Product();

tempProduct.RandomInit();

stack.Push(tempProduct);

isNextAdd= true;

break;

//Добавление элемента типа Goods

case 2:

Goods tempGoods = new Goods();

tempGoods.RandomInit();

isNextAdd= true;

stack.Push(tempGoods);

break;

//Добавление элемента типа MilkProduct

case 3:

MilkProduct tempMilkProduct = new MilkProduct();

tempMilkProduct.RandomInit();

stack.Push(tempMilkProduct);

isNextAdd= true;

break;

//Добавление элемента типа Toy

case 4:

Toy tempToy = new Toy();

tempToy.RandomInit();

stack.Push(tempToy);

isNextAdd= true;

break;

default:

Console.Clear();

break;

}

} while (!isNextAdd);

Console.Clear();

break;

case 0:

isNext=true;

break;

default:

Console.Clear();

break;

}

Console.Clear();

} while (!isNext);

isNext = false;

do

{

//Вывод стэка

Console.WriteLine("Стэк:");

if (stack.Count == 0)

{

Console.WriteLine("Пусто");

}

foreach (var item in stack)

{

item.Show();

}

Console.WriteLine($@"

Запросы для Stack

[1] - Посчитать количество элементов типа Toy

[2] - Распечатать элементы типа Product

[3] - Посчитать общую стоимость элементов типа MilkProduct

[0] - Дальше");

input = InputNumberMenu();

Console.Clear();

switch (input)

{

//Подсчёт количества элементов типа Toy

case 1:

int count = 0;

foreach (var item in stack)

{

if (item is Toy)

{

count++;

}

}

Console.ForegroundColor=ConsoleColor.Red;

Console.WriteLine("Количество элементов типа Toy: "+count);

Console.ForegroundColor=ConsoleColor.White;

break;

//Печачть элементов типа Product

case 2:

Console.ForegroundColor=ConsoleColor.Red;

foreach (var item in stack)

{

if (item is Product)

{

item.Show();

}

}

Console.ForegroundColor=ConsoleColor.White;

break;

//Нахождение общей стоимости элементов типа MilkProduct

case 3:

int summary = 0;

foreach (var item in stack)

{

if (item is MilkProduct)

{

summary += ((MilkProduct)item).TotalCost();

}

}

Console.ForegroundColor=ConsoleColor.Red;

Console.WriteLine("Общая стоимость элементов типа MilkProduct: "+summary);

Console.ForegroundColor=ConsoleColor.White;

break;

case 0:

isNext=true;

break;

default:

Console.Clear();

break;

}

} while (!isNext);

//Вывод стэка

Console.Clear();

Console.WriteLine("Стэк:");

if (stack.Count == 0)

{

Console.WriteLine("Пусто");

}

foreach (var item in stack)

{

item.Show();

}

Console.WriteLine("");

//Создание стэка для клонирования

Stack<Goods> clonnedStack = new Stack<Goods>(stack.Count);

int countStack = stack.Count;

Goods[] tempArray = new Goods[countStack];

stack.CopyTo(tempArray, 0);

Stack<Goods> tempStack = new Stack<Goods>(stack.Count);

//Перенос элементов начального стэка во временный

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

{

tempStack.Push(tempArray[i]);

}

//Перенос из временного стэка в новый для сохранения порядка элементов

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

{

clonnedStack.Push(tempStack.Pop());

}

Console.WriteLine("");

//Печать элементов "Клонированного" стэка

foreach (var item in clonnedStack)

{

item.Show();

}

Console.WriteLine(@"

Введите любой символ, чтобы продолжить.");

Console.ReadLine();

Console.Clear();

//Инициализация пустого стэка для сортировки

Stack<Goods> sortedStack = new Stack<Goods>(clonnedStack.Count);

//Сортировка стэка

sortedStack = SortStack(clonnedStack);

Console.WriteLine(@"Отсортированный стэк:");

//Печать сортированного стэка

foreach (var item in sortedStack)

{

item.Show();

}

Console.WriteLine(@"

Введите любой символ, чтобы продолжить.");

Console.ReadLine();

Console.Clear();

// Поиск элементов с количеством от 25 до 50

Console.WriteLine(@"Найденные элементы с количеством от 25 до 50:

");

//Вывод элементов со значением количества(Amount) в пределах от 25 до 50(включая)

foreach (var item in sortedStack)

{

if (item.Amount<=50 && item.Amount>=25)

{

item.Show();

}

}

Console.WriteLine(@"

Введите любой символ, чтобы продолжить.");

Console.ReadLine();

Console.Clear();

//Часть 2

//Инициализация объекто для создания списка(List), состоящего из объектов класса Goods

goods1.RandomInit();

goods2.RandomInit();

toy1.RandomInit();

toy2.RandomInit();

milkProduct1.RandomInit();

milkProduct2.RandomInit();

Goods[] goodsArrayPartTwo = { goods1, goods2, toy1, toy2, milkProduct1, milkProduct2 };

List<Goods> list = new List<Goods>(goodsArrayPartTwo);

isNext = false; ;

do // Вывод массива

{

Console.WriteLine("Список:");

if (list.Count == 0)

{

Console.WriteLine("Пусто");

}

foreach (var item in list)

{

item.Show();

Console.WriteLine("");

}

Console.WriteLine($@"

Меню для List

[1] - Удалить объект коллекции

[2] - Добавить случайный объект коллекции

[0] - Дальше");

input = InputNumberMenu();

switch (input)

{

// Удаление объекта коллекции по индексу

case 1:

Console.Clear();

Console.WriteLine("Введите позицию элемента, которое необходимо удалить:");

int index = InputNumber()-1;

if (list.Count>0 && index<list.Count && index>=0)

{

list.RemoveAt(index);

}

break;

//Добавления случайного объекта коллекции определённого типа

case 2:

Console.Clear();

bool isNextAdd = false;

do

{

Console.WriteLine(@"Какого класса добавить объект?

[1] - Product

[2] - Goods

[3] - MilkProduct

[4] - Toy");

input = InputNumberMenu();

Console.WriteLine("Введите позицию элемента, на которую будет помещён элемент:");

index = InputNumber();

if (list.Count>=0 && index<=list.Count+1 && index>=1)

{

switch (input)

{

//Добавление объекта типа Product

case 1:

Product tempProduct = new Product();

tempProduct.RandomInit();

list.Insert(index-1, tempProduct);

isNextAdd= true;

break;

//Добавление объекта типа Goods

case 2:

Goods tempGoods = new Goods();

tempGoods.RandomInit();

isNextAdd= true;

list.Insert(index-1, tempGoods);

break;

//Добавление объекта типа MilkProduct

case 3:

MilkProduct tempMilkProduct = new MilkProduct();

tempMilkProduct.RandomInit();

list.Insert(index-1, tempMilkProduct);

isNextAdd= true;

break;

//Добавление объекта типа Toy

case 4:

Toy tempToy = new Toy();

tempToy.RandomInit();

list.Insert(index - 1, tempToy);

isNextAdd= true;

break;

default:

Console.Clear();

break;

}

}

Console.Clear();

Console.ForegroundColor= ConsoleColor.Red;

Console.WriteLine("Неверное введено значение индекса!");

Console.ForegroundColor = ConsoleColor.White;

Console.WriteLine();

} while (!isNextAdd);

Console.Clear();

break;

case 0:

isNext=true;

break;

default:

Console.Clear();

break;

}

Console.Clear();

} while (!isNext);

// Запросы

Console.Clear();

isNext = false;

do

//Вывод списка

{

Console.WriteLine("Список:");

if (stack.Count == 0)

{

Console.WriteLine("Пусто");

}

foreach (var item in list)

{

item.Show();

Console.WriteLine("");

}

Console.WriteLine($@"

Запросы для List

[1] - Посчитать количество элементов типа Product

[2] - Распечатать элементы типа MilkProduct

[3] - Посчитать общую стоимость элементов типа Toy

[0] - Дальше");

input = InputNumberMenu();

Console.Clear();

switch (input)

{

//Запрос, считающий количество элементов типа Product в List

case 1:

int count = 0;

foreach (var item in list)

{

if (item is Product)

{

count++;

}

}

Console.Clear();

Console.ForegroundColor=ConsoleColor.Red;

Console.WriteLine("Количество элементов типа Product: "+count);

Console.ForegroundColor=ConsoleColor.White;

Console.WriteLine(@"

Введите любой символ, чтобы продолжить.");

Console.ReadLine();

Console.Clear();

break;

//Запрос, печатающий все элементы типа MilkProduct в List

case 2:

Console.Clear();

Console.ForegroundColor=ConsoleColor.Red;

foreach (var item in list)

{

if (item is MilkProduct)

{

item.Show();

Console.WriteLine("");

}

}

Console.ForegroundColor=ConsoleColor.White;

Console.WriteLine(@"

Введите любой символ, чтобы продолжить.");

Console.ReadLine();

Console.Clear();

break;

//Запрос, рассчитывающий общую стоимость товаров типа Toy в List

case 3:

int summary = 0;

foreach (var item in list)

{

if (item is Toy)

{

summary += ((Toy)item).TotalCost();

}

}

Console.Clear();

Console.ForegroundColor=ConsoleColor.Red;

Console.WriteLine("Общая стоимость элементов типа Toy: "+summary);

Console.ForegroundColor=ConsoleColor.White;

Console.WriteLine(@"

Введите любой символ, чтобы продолжить.");

Console.ReadLine();

Console.Clear();

break;

case 0:

isNext=true;

break;

default:

Console.Clear();

break;

}

} while (!isNext);

isNext = false;

//Сортировка списка List

list.Sort();

do

{

//Вывод отсортированного списка

Console.WriteLine("Отсортированный cписок:");

foreach (var item in list)

{

item.Show();

Console.WriteLine("");

}

Console.WriteLine("Введите количество продукции, которую вы собираетесь найти:");

int amount = InputNumber();

Console.Clear();

Goods foundGoods = new Goods(amount, 0);

int indexFind = 999;

//Бинарный поиск по индексу

indexFind = list.BinarySearch(foundGoods);

if (indexFind<0)

{

Console.WriteLine(@"Товар не найден!");

}

else

{

Console.WriteLine(@"Найденный продукт:

");

list[list.BinarySearch(foundGoods)].Show();

}

Console.WriteLine(@"

[Любое число, кроме 0] - Вернуться назад

[0] - конец");

input = InputNumberMenu();

if (input ==0)

{

isNext=true;

}

Console.Clear();

} while (!isNext);

//Создание клонированного списка типа Goods(List<Goods>)

List<Goods> clonnedList = new List<Goods>(list.Count);

int countList = list.Count;

Goods[] tempArrayList = new Goods[countList];

list.CopyTo(tempArrayList, 0);

List<Goods> tempList = new List<Goods>(list.Count);

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

{

clonnedList.Add(tempArrayList[i]);

}

Console.WriteLine("");

list.RemoveAt(0);

foreach (var item in clonnedList)

{

item.Show();

Console.WriteLine("");

}

}

[ExcludeFromCodeCoverage]

public static int InputNumber() // Ввод целого числа

{

bool ok;

int Num;

do

{

ok = int.TryParse(Console.ReadLine(), out Num);

if (!ok)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Вы ввели неверное значение!");

Console.ForegroundColor = ConsoleColor.White;

}

} while (!ok);

if (Num <= 0)

{

return 0;

}

else

{

return Num;

}

}

[ExcludeFromCodeCoverage]

public static int InputNumberMenu() // Ввод целого числа

{

bool ok;

int Num;

do

{

ok = int.TryParse(Console.ReadLine(), out Num);

if (!ok)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Вы ввели неверное значение!");

Console.ForegroundColor = ConsoleColor.White;

}

} while (!ok);

return Num;

}

[ExcludeFromCodeCoverage]

/// <summary>

/// Функция сортировка стэка

/// </summary>

/// <param name="input">Несортированный стэк</param>

/// <returns>отсортированный стэк</returns>

public static Stack<Goods> SortStack(Stack<Goods> input)

{

Stack<Goods> tempStack = new Stack<Goods>();

while (input.Count > 0)

{

//Перенос во временную переменную элемент несортированного стэка

Goods temp = input.Pop();

while (tempStack.Count > 0 && tempStack.Peek().Amount > temp.Amount)

{

input.Push(tempStack.Pop());

}

tempStack.Push(temp);

}

return tempStack;

}

}

}

# ПРИЛОЖЕНИЕ Б **Программа Program.cs(Часть 3)**

using ClassLibraryLab10;

using lab;

using System.Diagnostics.CodeAnalysis;

namespace \_11.\_3

{

internal class Program

{

[ExcludeFromCodeCoverage]

static void Main(string[] args)

{

// Часть 3

int size =10; //Начальный размер коллекций

TestCollections tc = new TestCollections(size); //Создание коллекций

//Пример добавления элементов в коллекции

tc.AddLinkedListOfToy();

tc.AddLinkedListOfString();

tc.AddDictionaryGoods();

tc.AddDictionaryString();

//Ожидание следюущей команды

DoNext();

Console.Clear();

//Вывод всех коллекций

tc.PrintAllCollections();

DoNext();

Console.Clear();

//Пример удаленимя элементов из коллекций

tc.RemoveLinkedListOfToy();

tc.RemoveLinkedListOfString();

tc.RemoveDictionaryGoods();

tc.RemoveDictionaryString();

//Вывод всех коллекций

tc.PrintAllCollections();

//Ожидание следующей команды

DoNext();

Console.Clear();

//Поиск первого, среднего и последнего элемента в коллекциях

tc.FindAllElements();

}

[ExcludeFromCodeCoverage]

// Функция ожидания нажатия клавиши

public static void DoNext()

{

Console.ForegroundColor= ConsoleColor.Green;

Console.WriteLine("Нажмите Enter, чтобы продолжить");

do

{

} while (Console.ReadKey().Key!=ConsoleKey.Enter);

Console.ForegroundColor= ConsoleColor.White;

}

}

}

# ПРИЛОЖЕНИЕ В **Программа Goods.cs**

using ClassLibraryLab10;

using System;

using System.Collections.Generic;

using System.Diagnostics.CodeAnalysis;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading.Tasks;

namespace lab

{

public class Goods:IInit, IComparable, ICloneable

{

protected int amount = 0;

protected Random random = new Random();

public int Amount

{

get => amount;

set

{

if (amount>=0)

{

amount = value;

}

else

{

amount=0;

}

}

}

protected int price = 0;

public int Price

{

get => price;

set

{

if (price>=0)

{

price = value;

}

else

{

price=0;

}

}

}

[ExcludeFromCodeCoverage]

public virtual void Init() // Инициализация цены и количества

{

Console.WriteLine("Введите количество товара:");

Amount = EnterNumber();

Console.WriteLine("Введите цену товара:");

Price = EnterNumber();

}

[ExcludeFromCodeCoverage]

public virtual void RandomInit() // Инициализация цены и количества с помощью ДСЧ

{

Amount = random.Next(1, 1001);

Price = random.Next(1, 1001);

}

public Goods()

{

Amount = 0;

Price = 0;

}

[ExcludeFromCodeCoverage]

public override int GetHashCode()

{

return Amount.GetHashCode()+Price.GetHashCode() ;

}

public Goods (int amount,int price)

{

Amount = amount;

Price = price;

}

[ExcludeFromCodeCoverage]

public virtual void Show() //Вывод цены и количества

{

Console.WriteLine($"Количество товара: {Amount}, Стоимость товара: {Price} руб.");

}

public override bool Equals(object obj) //Перегрузка метода сравнения для тестов

{

if (obj is Goods goods)

{

return ((goods.Amount == this.Amount)&&(goods.Price == this.Price));

}

else

{

return false;

}

}

[ExcludeFromCodeCoverage]

public static int EnterNumber() // Ввод целого числа

{

bool ok;

int Num;

do

{

ok = int.TryParse(Console.ReadLine(), out Num);

if (!ok)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Вы ввели неверное значение!");

Console.ForegroundColor = ConsoleColor.White;

}

} while (!ok);

if (Num <= 0)

{

return 0;

}

else

{

return Num;

}

}

[ExcludeFromCodeCoverage]

public int CompareTo(object? obj)

{

if (!(obj is Goods)) return -1;

Goods goods = (Goods)obj;

return Amount.CompareTo(goods.Amount);

}

[ExcludeFromCodeCoverage]

public object ShallowCopy()

{

return this.MemberwiseClone();

}

public object Clone()

{

return new Goods(Amount, Price);

}

[ExcludeFromCodeCoverage]

public override string ToString()

{

return ($"Количество товара: {Amount}, Стоимость товара: {Price}.");

}

}

}

# ПРИЛОЖЕНИЕ Г **Программа Product.cs**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Diagnostics.CodeAnalysis;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab

{

public class Product: Goods

{

protected Random random = new Random();

[ExcludeFromCodeCoverage]

public override void Init()

{

base.Init();

Console.WriteLine("Введите название товара:");

Name = Console.ReadLine();

if (Name =="")

{

Name ="NoName";

}

else

{

Name = Name;

}

}

[ExcludeFromCodeCoverage]

public override void RandomInit()

{

base.RandomInit();

string[] nameArray = new string[] { "Лук Репчатый", "Jack Daniels", "Томатная паста", "Рожки", "Покровский хлеб", "Горошек зелёный", "Питахайя", "Мороженое Baskin Robbins", "Яблоки гольден","Кола" };

Name = nameArray[random.Next(nameArray.Length)];

}

public string Name { get; set; }

public Product()

: base()

{

Name ="NoName";

}

public Product(int amount, int price, string name)

: base(amount, price)

{

Name = name;

}

[ExcludeFromCodeCoverage]

public override void Show()

{

base.Show();

Console.WriteLine($"Наименование - {Name}");

}

[ExcludeFromCodeCoverage]

public new void ShowName()

{

Console.WriteLine($"Наименование - {Name}");

}

//Метод переопределения функции(overload) работает некорректно, так как в массиве будет вызываться элемент класса Goods, а не каждый отдельно

//public new void Show()

//{

// base.Show();

// Console.WriteLine($"Наименование - {Name}");

//}

public int TotalCost()

{

return Amount\*Price;

}

public override bool Equals(object obj)

{

if (obj is Product product)

{

return ((this.Price == product.Price)&&(this.Amount == product.Amount)&&(string.Compare(this.Name, product.Name)==0));

}

else

{

return false;

}

}

}

}

# ПРИЛОЖЕНИЕ Д **Программа MilkProduct.cs**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Diagnostics.CodeAnalysis;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab

{

public class MilkProduct: Goods

{

protected Random random = new Random();

protected int fatPercent = 0;

public int FatPercent

{

get => fatPercent;

set

{

if (fatPercent>=0)

{

fatPercent=value;

}

else

{

fatPercent=0;

}

}

}

[ExcludeFromCodeCoverage]

public override void Init()

{

base.Init();

Console.WriteLine("Введите название товара:");

Name = Console.ReadLine();

if (Name =="")

{

Name ="NoName";

}

else

{

Name = Name;

}

Console.WriteLine("Введите жирность продукта:");

FatPercent = EnterFatPercent();

}

[ExcludeFromCodeCoverage]

public override void RandomInit()

{

base.RandomInit();

string[] nameArray = new string[] { "Сырок Б. Ю. Александров","Молоко из Нытвы","Талицкий йогурт","Простокваша","Сметана","Молочный коктейль Чудо","Кумыс","Айран","Творог","Биолайф"};

Name = nameArray[random.Next(nameArray.Length)];

FatPercent = random.Next(0,26);

}

public string Name { get; set; }

public MilkProduct()

: base()

{

Name = "NoName";

}

public MilkProduct(int amount, int price, string name, int fatPercent)

: base(amount, price)

{

Name = name;

FatPercent = fatPercent;

}

[ExcludeFromCodeCoverage]

public override void Show()// Вывод количество товара, его цены и имя

{

base.Show();

Console.WriteLine($"Наименование - {Name}");

Console.WriteLine($"Жирность продукта - {FatPercent}%");

}

[ExcludeFromCodeCoverage]

public new void ShowName() //Вывод наименования товара

{

Console.WriteLine($"Наименование - {Name}");

}

//Метод переопределения функции(overload) работает некорректно, так как в массиве будет вызываться элемент класса Goods, а не каждый отдельно

//public new void Show()

//{

// base.Show();

// Console.WriteLine($"Наименование - {Name}");

//}

public int TotalCost() // Нахождение общей стоимости

{

return Amount\*Price;

}

public override bool Equals(object obj) //Перегрузка метода сравнения

{

if (obj is MilkProduct milkProduct)

{

return ((this.Price == milkProduct.Price)&&(this.Amount == milkProduct.Amount)&&(string.Compare(this.Name, milkProduct.Name)==0));

}

else

{

return false;

}

}

[ExcludeFromCodeCoverage]

public static int EnterFatPercent() // Ввод целого числа

{

bool ok;

int Num;

do

{

ok = int.TryParse(Console.ReadLine(), out Num);

if ((!ok)||(Num>100))

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("Вы ввели неверное значение!");

Console.ForegroundColor = ConsoleColor.White;

}

} while (!(ok&& (Num<=100)));

if (Num <= 0)

{

return 0;

}

else

{

return Num;

}

}

}

}

# ПРИЛОЖЕНИЕ Е **Программа Toy.cs**

using System;

using System.Collections.Generic;

using System.Diagnostics.CodeAnalysis;

using System.Linq;

using System.Security.Cryptography.X509Certificates;

using System.Text;

using System.Threading.Tasks;

namespace lab

{

public class Toy:Goods

{

public string Name;

protected Random random = new Random();

[ExcludeFromCodeCoverage]

public override void Init()

{

base.Init();

Console.WriteLine("Введите название товара:");

Name = Console.ReadLine();

if (Name =="")

{

Name ="NoName";

}

else

{

Name = Name;

}

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

ToyType = Console.ReadLine();

if ( ToyType== "")

{

ToyType = "NoType";

}

else

{

ToyType = ToyType;

}

}

[ExcludeFromCodeCoverage]

public override void RandomInit()

{

base.RandomInit();

string[] descriptionArray = new string[] { "Гигантская игрушка ", "Маленькая игрушка ", "Мягкая игрушка ", "Детская игрушка ", "Семейная игрушка ", "Высокоточная игрушка ", "Низкая игрушка ", "Многофункциональная игрушка ", "Музыкальная игрушка ", "Одноцветная игрушка " };

string[] nameArray = new string[] { "Хагги Вагги", "Амонг Ус", "Бравл Старс","Акула из икеи","Капибара","Кукла вуду","Утка-обнимашка","Базз Лайтер","Спиннер","Поп-Ит","кукла", "мяч", "конструктор", "пазл", "машинка", "кубики", "пистолет", "каталка", "игровой набор", "плюшевый мишка", "раскраски", "домик для кукол", "богомол","игровой коврик", "пластилин" };

string[] ageArray = new string[] { " 0+", " 3+", " 6+", " 12+", " 14+", " 16+", " 18+", " 21+", " 45+", " 99+" };

Name = descriptionArray[random.Next(descriptionArray.Length)] + nameArray[random.Next(nameArray.Length)] + ageArray[random.Next(ageArray.Length)];

base.RandomInit();

string[] typeArray = new string[] { "Мягкая игрушка", "Конструктор", "Головоломка", "Пазлы", "Модель" };

ToyType = typeArray[random.Next(typeArray.Length)];

}

public string ToyType { get;set; }

public Toy()

: base()

{

Name = "NoName";

ToyType = "NoType";

}

public Goods GetBase()

{

return new Goods(Amount, Price);

}

public Toy(int amount, int price, string name, string toyType)

:base (amount, price)

{

Name = name;

ToyType = toyType;

}

[ExcludeFromCodeCoverage]

public override void Show()

{

base.Show();

Console.WriteLine($"Наименование - {Name}");

Console.WriteLine($"Тип игрушки - {ToyType}");

}

[ExcludeFromCodeCoverage]

public new void ShowName()

{

Console.WriteLine($"Наименование - {Name}");

}

//Метод переопределения функции(overload) работает некорректно, так как в массиве будет вызываться элемент класса Goods, а не каждый отдельно

//public new void Show()

//{

// base.Show();

// Console.WriteLine($"Наименование - {Name}");

//}

[ExcludeFromCodeCoverage]

public override string ToString()

{

return ($@"Количество товара: {Amount}, Стоимость товара: {Price}.

Наименование - {Name}

Тип игрушки - {ToyType}");

}

public int TotalCost()

{

return Amount\*Price;

}

public override bool Equals(object obj)

{

if (obj is Toy toy)

{

return ((this.Price == toy.Price)&&(this.Amount == toy.Amount)&&(string.Compare(this.Name, toy.Name)==0));

}

else

{

return false;

}

}

}

}

# ПРИЛОЖЕНИЕ Ё **Программа Animal.cs**

using lab;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Diagnostics.CodeAnalysis;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibraryLab10

{

public class Animal:IInit

{

static Random random = new Random();

static string[] Names = { "Лиса", "Слон", "Лев", "Капибара", "Аксолотль", "Мурена", "Крокодил" };

public string Name { get; set; }

int age;

public int Age

{

get => age;

set

{

if (age>=0 && age<=50)

{

age=value;

}

else

{

age=0;

}

}

}

public Animal()

{

Age = 1;

Name = "NoName";

}

public Animal(string name, int age)

{

Name=name;

Age=age;

}

[ExcludeFromCodeCoverage]

public void Show()

{

Console.WriteLine($"Название животного: {Name}, Возраст: {Age}");

}

[ExcludeFromCodeCoverage]

public virtual void Init()

{

Console.WriteLine("Введите название животного:");

Name = Console.ReadLine();

Console.WriteLine("Введите возраст:");

Age = EnterAge();

}

[ExcludeFromCodeCoverage]

public virtual void RandomInit()

{

Name = Names[random.Next(Names.Length)];

Age = random.Next(1, 50);

}

[ExcludeFromCodeCoverage]

public static int EnterAge()

{

int Number;

bool ok;

do

{

ok = ((int.TryParse(Console.ReadLine(), out Number))&&(Number >0)&&(Number <51));

if (!ok)

{

Console.WriteLine("Вы ввели неверное значение!");

}

} while (!ok);

return Number;

}

public override bool Equals(object obj)

{

if (obj is Animal animal)

{

return ((this.Age == animal.Age)&&(string.Compare(this.Name, animal.Name)==0));

}

else

{

return false;

}

}

}

}

# ПРИЛОЖЕНИЕ Ж **Программа SortByPrice.cs**

using lab;

using System;

using System.Collections;

using System.Diagnostics.CodeAnalysis;

using System.Linq;

using System.Runtime.InteropServices.ObjectiveC;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibraryLab10

{

[ExcludeFromCodeCoverage]

public class SortByPrice:IComparer

{

public int Compare(object x, Object y)

{

Goods goods1 = x as Goods;

Goods goods2 = y as Goods;

if (goods1.Price<goods2.Price)

{

return -1;

}

else

{

if (goods2.Price==goods1.Price)

{

return 0;

}

else

{

return 1;

}

}

}

}

}

# ПРИЛОЖЕНИЕ З **Программа IInit.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Security.Cryptography.X509Certificates;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibraryLab10

{

public interface IInit

{

void RandomInit();

void Init();

void Show();

}

}

# ПРИЛОЖЕНИЕ И **Программа TestCollections.cs**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Diagnostics.CodeAnalysis;

using System.Drawing;

using System.Linq;

using System.Reflection.Metadata;

using System.Security.Cryptography.X509Certificates;

using System.Text;

using System.Threading.Tasks;

using ClassLibraryLab10;

namespace lab

{

class TestCollections

{

// Создание пустых коллекций

LinkedList<Toy> linkedListToy = new LinkedList<Toy>();

LinkedList<string> linkedListString = new LinkedList<string>();

Dictionary<Goods, Toy> dictionaryGoods = new Dictionary<Goods, Toy>();

Dictionary<string, Toy> dictionaryString = new Dictionary<string, Toy>();

// Создание пустых объектов для первого, среднего, последнего и несуществующего объектов типа Toy

Toy first, middle, last;

Toy notFound = new Toy(0, 0, "noName", "NoType");

[ExcludeFromCodeCoverage]

/// <summary>

/// Функция инициализации коллекций с заданным начальным размером

/// </summary>

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

void InitCollections(int size)

{

// Цикл инициализации объектов коллекции

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

{

//Обработка исключения для создания уникальности коллекций

try

{

Toy tempToy = new Toy();

tempToy.RandomInit();

Goods tempGoods = new Goods();

tempGoods.Amount=tempToy.Amount;

tempGoods.Price=tempToy.Price;

linkedListToy.AddLast(tempToy);

linkedListString.AddLast(tempToy.ToString());

dictionaryGoods.Add(tempGoods, tempToy);

dictionaryString.Add(tempGoods.ToString(), tempToy);

if (i==0)

{

first= new Toy (tempToy.Amount, tempToy.Price, tempToy.Name, tempToy.ToyType);

}

if (i == size - 1)

{

last = new Toy(tempToy.Amount, tempToy.Price, tempToy.Name, tempToy.ToyType);

}

if (i == size/2)

{

middle = new Toy(tempToy.Amount, tempToy.Price, tempToy.Name, tempToy.ToyType);

}

}

catch (Exception)

{

i--;

}

}

}

[ExcludeFromCodeCoverage]

public void PrintLinkedListOfToy(LinkedList<Toy> collection)

{

Console.WriteLine("Печать коллекции LinkedList<Toy>:");

foreach(Toy item in collection)

{

item.Show();

}

}

[ExcludeFromCodeCoverage]

public void PrintLinkedListOfString(LinkedList<string> collection)

{

Console.WriteLine("Печать коллекции LinkedList<string>:");

foreach (string item in collection)

{

Console.WriteLine(item);

}

}

[ExcludeFromCodeCoverage]

public void PrintDictionaryOfGoods(Dictionary<Goods,Toy> dictionary)

{

Console.WriteLine("Печать коллекции Dicitonary<Goods, Toy>:");

ICollection<Goods> keys = dictionary.Keys;

foreach (Goods item in keys)

{

Console.Write(item.ToString() + " | ");

dictionary[item].Show();

}

}

[ExcludeFromCodeCoverage]

public void PrintDictionaryOfString(Dictionary<string, Toy> dictionary)

{

Console.WriteLine("Печать коллекции Dicitonary<string, Toy>:");

ICollection<string> keys = dictionary.Keys;

foreach (string item in keys)

{

Console.WriteLine($"{item.ToString()} | {dictionary[item].ToString()}");

}

}

[ExcludeFromCodeCoverage]

public TestCollections(int size)

{

InitCollections(size);

PrintLinkedListOfToy(linkedListToy);

Console.WriteLine();

PrintLinkedListOfString(linkedListString);

Console.WriteLine();

PrintDictionaryOfGoods(dictionaryGoods);

Console.WriteLine();

PrintDictionaryOfString(dictionaryString);

}

[ExcludeFromCodeCoverage]

/// <summary>

/// Функция добавления элемента к коллекции LinkedList<Toy>

/// </summary>

public void AddLinkedListOfToy()

{

Toy tempToy = new Toy();

tempToy.RandomInit();

try

{

linkedListToy.AddLast(tempToy);

}

catch(Exception)

{

Console.WriteLine("Данный элемент уже существует в коллекции!");

}

}

[ExcludeFromCodeCoverage]

/// <summary>

/// Функция добавления элемента к коллекции LinkedList<string>

/// </summary>

public void AddLinkedListOfString()

{

Toy tempToy = new Toy();

tempToy.RandomInit();

try

{

linkedListString.AddLast(tempToy.ToString());

}

catch (Exception)

{

Console.WriteLine("Данный элемент уже существует в коллекции!");

}

}

[ExcludeFromCodeCoverage]

/// <summary>

/// Функция добавления элемента к Dictionary<Goods, Toy>

/// </summary>

public void AddDictionaryGoods()

{

Toy tempToy = new Toy();

tempToy.RandomInit();

Goods tempGoods = new Goods();

tempGoods.Amount=tempToy.Amount;

tempGoods.Price=tempToy.Price;

try

{

dictionaryGoods.Add(tempGoods, tempToy);

}

catch (Exception)

{

Console.WriteLine("Данный элемент уже существует в коллекции!");

}

}

[ExcludeFromCodeCoverage]

/// <summary>

/// Функция добавления элемента к Dictionary<string, Toy>

/// </summary>

public void AddDictionaryString()

{

Toy tempToy = new Toy();

tempToy.RandomInit();

Goods tempGoods = new Goods();

tempGoods.Amount=tempToy.Amount;

tempGoods.Price=tempToy.Price;

try

{

dictionaryString.Add(tempGoods.ToString(), tempToy);

}

catch (Exception)

{

Console.WriteLine("Данный элемент уже существует в коллекции!");

}

}

[ExcludeFromCodeCoverage]

public void PrintAllCollections()

{

PrintLinkedListOfToy(linkedListToy);

Console.WriteLine();

PrintLinkedListOfString(linkedListString);

Console.WriteLine();

PrintDictionaryOfGoods(dictionaryGoods);

Console.WriteLine();

PrintDictionaryOfString(dictionaryString);

}

[ExcludeFromCodeCoverage]

/// <summary>

/// Функция удаления элемента из коллекции LinkedList<Toy>

/// </summary>

public void RemoveLinkedListOfToy()

{

try

{

linkedListToy.RemoveLast();

}

catch (Exception)

{

Console.WriteLine("Невозможно удалить элемент!");

}

}

[ExcludeFromCodeCoverage]

/// <summary>

/// Функция удаления элемента из коллекции LinkedList<String>

/// </summary>

public void RemoveLinkedListOfString()

{

try

{

linkedListString.RemoveLast();

}

catch (Exception)

{

Console.WriteLine("Невозможно удалить элемент!");

}

}

[ExcludeFromCodeCoverage]

/// <summary>

/// Функция удаления элемента из коллекции Dictionary<Goods, Toy>

/// </summary>

public void RemoveDictionaryGoods()

{

try

{

dictionaryGoods.Remove(dictionaryGoods.Keys.Last());

}

catch (Exception)

{

Console.WriteLine("Невозможно удалить элемент!");

}

}

[ExcludeFromCodeCoverage]

/// <summary>

/// Функция удаления элемента из коллекции Dictionary<string, Toy>

/// </summary>

public void RemoveDictionaryString()

{

try

{

dictionaryString.Remove(dictionaryString.Keys.Last());

}

catch (Exception)

{

Console.WriteLine("Невозможно удалить элемент!");

}

}

[ExcludeFromCodeCoverage]

public void FindAllElements()

{

//Нахождение времени для первого элемента

FindElement(first);

//Нахождение времени для среднего элемента

FindElement(middle);

//Нахождение времени для последнего элемента

FindElement(last);

//Нахождение времени для несуществующего элемента

FindElement(notFound);

}

[ExcludeFromCodeCoverage]

public void FindElement(Toy element)

{

//Нахождение элемента в коллекции LinkedList<Toy>

Stopwatch timer = new Stopwatch();

timer.Restart();

bool ok = linkedListToy.Contains(element);

timer.Stop();

if (ok)

{

Console.WriteLine($"Элемент найден в LinkedList<Toy> за {timer.ElapsedTicks}");

}

else {

Console.WriteLine("Элемент не был найден в LinkedList<Toy>!");

}

//Нахождение элемента в коллекции LinkedList<string>

timer.Restart();

ok = linkedListString.Contains(element.ToString());

timer.Stop();

if (ok)

{

Console.WriteLine($"Элемент найден в LinkedList<string> за {timer.ElapsedTicks}");

}

else

{

Console.WriteLine("Элемент не был найден в LinkedList<string>!");

}

//Нахождение элемента в коллекции Dictionary<Goods, Toy>

timer.Restart();

ok = dictionaryGoods.ContainsKey(element.GetBase());

timer.Stop();

if (ok)

{

Console.WriteLine($"Элемент найден в Dictionary<Goods, Toy> за {timer.ElapsedTicks}");

}

else

{

Console.WriteLine("Элемент не был найден в Dictionary<Goods, Toy>!");

}

//Нахождение элемента в коллекции LinkedList<string, Toy>

timer.Restart();

ok = dictionaryString.ContainsKey(element.GetBase().ToString());

timer.Stop();

if (ok)

{

Console.WriteLine($"Элемент найден в Dictionary<string, Toy> за {timer.ElapsedTicks}");

}

else

{

Console.WriteLine("Элемент не был найден в Dictionary<string, Toy>!");

}

}

}

}