# Практическая работа № 11. Коллекции.

**Цель**. Получить практические навыки работы со стандартными коллекциями пространства имен System.Collection.

* Получить практические навыки работы со стандартными параметризированными коллекциями пространства имен System.Collection.Generic.

## **2. Постановка задачи**

### **2.1. Задание 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.

При работе с коллекцией использовать объекты из иерархии классов, разработанной в работе №10.

(

### **2.2. Задание 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.

При работе с коллекцией использовать объекты из иерархии классов, разработанной в работе №10.

### **2.3. Задание 3.**

1. Создать иерархию классов (базовый – производный) в соответствии с вариантом (см. лаб. раб. №10).
2. В производном классе определить свойство, которое возвращает ссылку на объект базового класса (это свойство должно возвращать ссылку на объект базового класса, а не ссылку на вызывающий объект производного класса).
3. Определить класс TestCollections, который содержит поля следующих типов

Коллекция\_1<TKey> ;

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.

1. Написать конструктор класса TestCollections, в котором создаются коллекции с заданным числом элементов.
2. Предусмотреть автоматическую генерацию элементов коллекции таким образом, что каждый объект (Student) содержит подобъект базового класса (Person).
3. Все четыре коллекции должны содержать одинаковое число элементов. Каждому элементу из коллекции Коллекция\_1<TKey> должен отвечать элемент в коллекции Коллекция\_2<TKey, TValue> с равным значением ключа. Список Коллекция\_1<string> состоит из строк, которые получены в результате вызова метода ToString() для объектов TKey из списка Коллекция\_1<TKey>. Каждому элементу списка Коллекция\_1<string> отвечает элемент в Коллекция\_2 <string, TValue> с равным значением ключа типа string.
4. Для четырех разных элементов – первого, центрального, последнего и элемента, не входящего в коллекцию – надо измерить время поиска элемента в коллекциях Коллекция\_1<TKey> 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!
5. Предусмотреть методы для работы с коллекциями (добавление и удаление элементов).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № варианта | Задание 1 | Задание 2 | Задание 3 (Коллекция\_1) | Задание 3 (Коллекция\_2) |
| 7 | Stack | List<T> | LinkedList <T> | Dictionary <K,T> |

## **4. Содержание отчета**

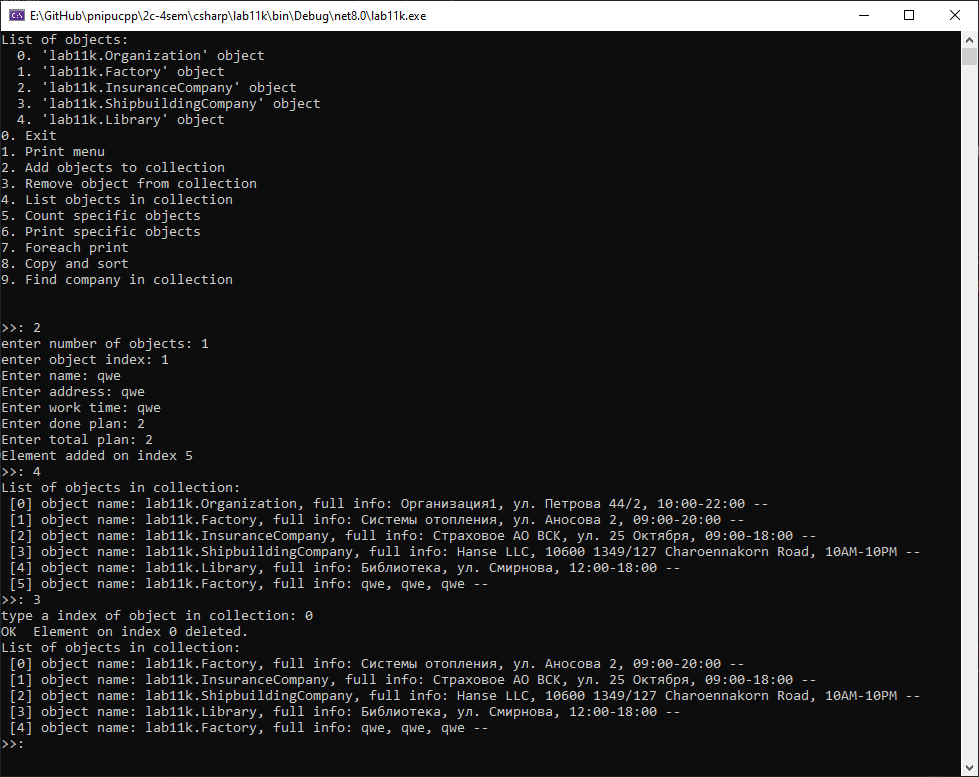
**Для заданий 11.1 и 11.2**

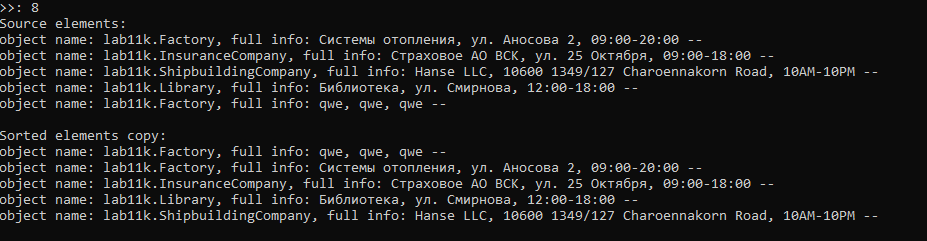
1. Диаграмма классов (в том числе иерархия классов для объектов, помещаемых в коллекцию).
2. Описание методов, используемых для работы с коллекцией.
3. Пример демонстрационной программы.

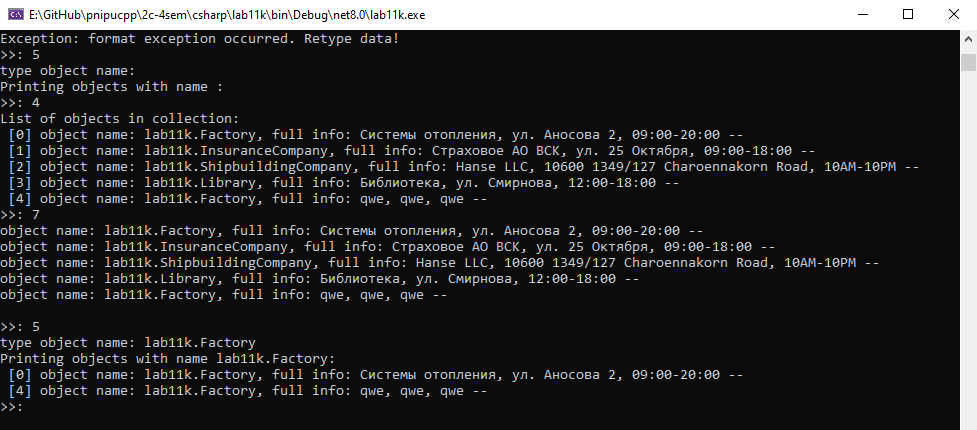
**Для задания 11.3**

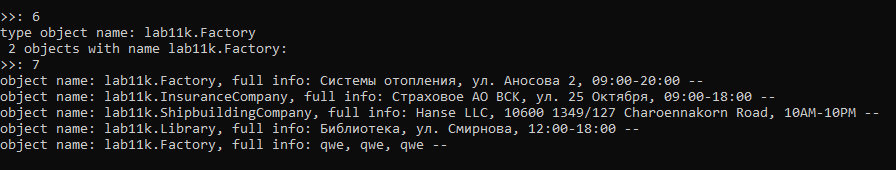
1. Диаграмма классов (в том числе иерархия классов для объектов, помещаемых в коллекцию).
2. Пример демонстрационной программы.
3. Полученные результаты.
4. Анализ полученных результатов (объяснить, какая из рассмотренных коллекций оказалась самой быстродействующей, и почему).

**Задание 1**

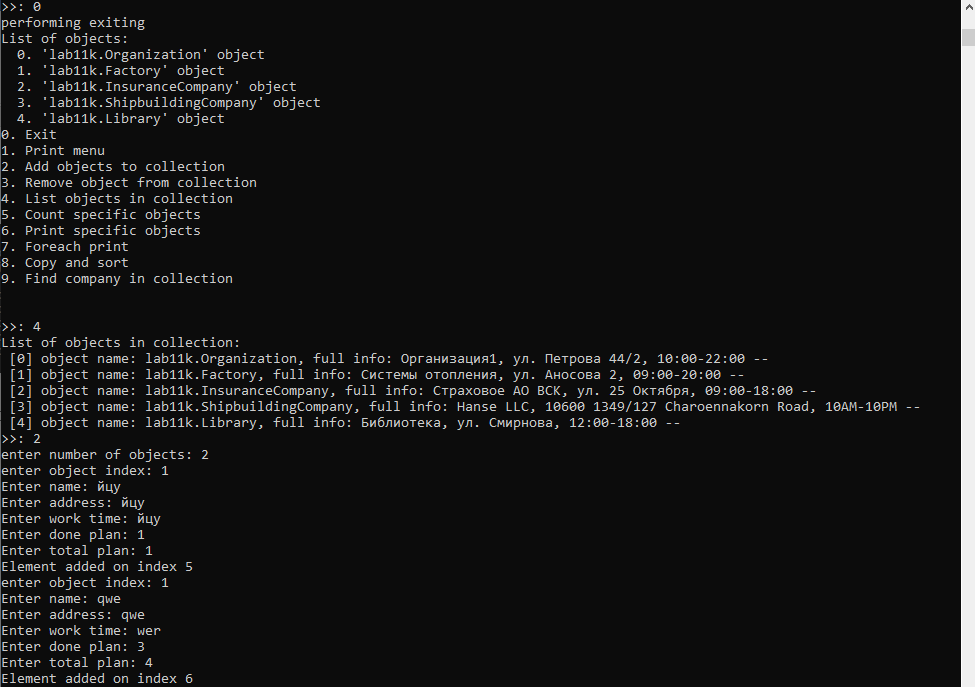


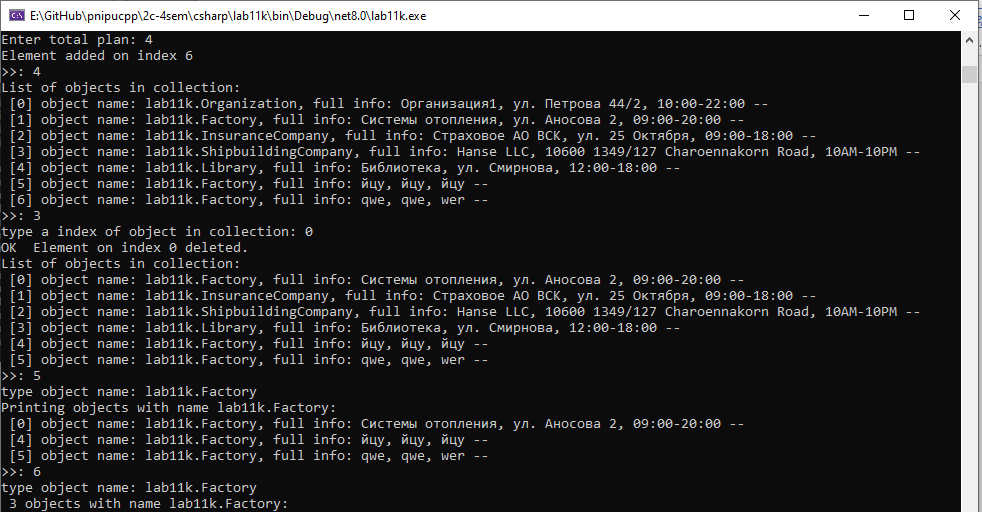


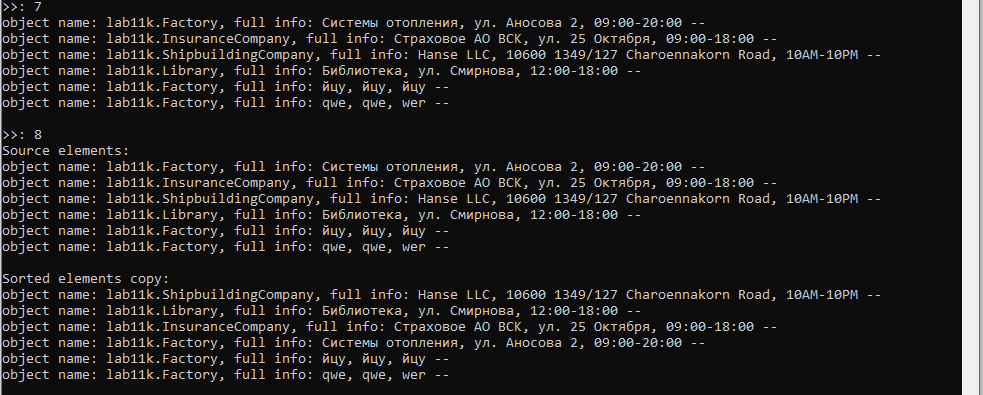




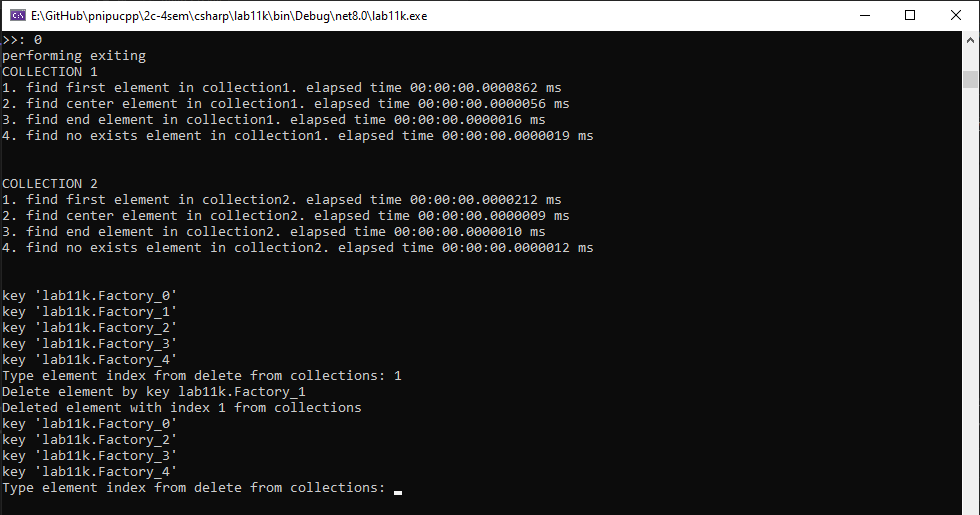
**Задание 2**







**Задание 3**



**Исходный код**

**CollectionsLab1.cs**

namespace lab11k

{

public class CollectionsLab1 : ConInFormat

{

private Stack<Organization> organizations;

public CollectionsLab1()

{

organizations = new Stack<Organization>();

}

private enum ORG\_TYPE

{

ORGANIZATION = 0,

FACTORY,

INSURANCE\_COMPANY,

SHIPBUILDING\_COMPANY,

LIBRARY

};

protected void InsertDefaultOrgs()

{

organizations.Push(new Organization("Организация1", "ул. Петрова 44/2", "10:00-22:00"));

organizations.Push(new Factory("Системы отопления", "ул. Аносова 2", "09:00-20:00", 12000, 10000));

organizations.Push(new InsuranceCompany("Страховое АО ВСК", "ул. 25 Октября", "09:00–18:00", 254003));

organizations.Push(new ShipbuildingCompany("Hanse LLC", "10600 1349/127 Charoennakorn Road", "10AM-10PM", 18000, 6000, 24000, 2400));

organizations.Push(new Library("Библиотека", "ул. Смирнова", "12:00-18:00", 1400));

}

void PrintOrganizationObjects()

{

object[] objects = GetObjects();

Console.WriteLine("List of objects:");

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

Console.WriteLine(" {0}. '{1}' object", i, objects[i].ToString());

}

Organization CreateOrganization(ORG\_TYPE type)

{

Organization orgobj;

switch (type)

{

default:

case ORG\_TYPE.ORGANIZATION:

orgobj = new Organization();

break;

case ORG\_TYPE.FACTORY:

orgobj = new Factory();

break;

case ORG\_TYPE.INSURANCE\_COMPANY:

orgobj = new InsuranceCompany();

break;

case ORG\_TYPE.SHIPBUILDING\_COMPANY:

orgobj = new ShipbuildingCompany();

break;

case ORG\_TYPE.LIBRARY:

orgobj = new Library();

break;

}

orgobj.FillInfo();

return orgobj;

}

private void PrintMenu()

{

Console.WriteLine(

"0. Exit\n" +

"1. Print menu\n" +

"2. Add objects to collection\n" +

"3. Remove object from collection\n" +

"4. List objects in collection\n" +

"5. Count specific objects\n" +

"6. Print specific objects\n" +

"7. Foreach print\n" +

"8. Copy and sort\n" +

"9. Find company in collection\n\n"

);

}

private void PrintObjectsInCollection()

{

Console.WriteLine("List of objects in collection:");

Organization[] arr = organizations.ToArray();

Array.Reverse(arr);

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

{

Console.WriteLine(" [{0}] object name: {1}, full info: {2}", i, arr[i].ToString(), arr[i].GetFullInfo());

}

}

private void InsertNewOrgs()

{

/\* add new organizations to collection \*/

int count = (int)ReadLineWithDescription(FORMAT\_TYPE.FORMAT\_INT32, "enter number of objects");

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

{

int orgtype = (int)ReadLineWithDescription(FORMAT\_TYPE.FORMAT\_INT32, "enter object index");

Organization org = CreateOrganization((ORG\_TYPE)orgtype);

organizations.Push(org);

Console.WriteLine("Element added on index {0}", organizations.Count - 1);

}

}

private object[] GetObjects()

{

object[] objects = {

new Organization(),

new Factory(),

new InsuranceCompany(),

new ShipbuildingCompany(),

new Library()

};

return objects;

}

public Stack<T> SortStack<T>(Stack<T> stack) where T : IComparable<T>

{

T[] elements = stack.ToArray();

Array.Sort(elements);

Array.Reverse(elements);

return new Stack<T>(elements);

}

private bool FindObjectByName(ref object dst, string? name)

{

object[] objects = GetObjects();

foreach (object obj in objects)

{

if (name == obj.ToString())

{

dst = obj;

return true;

}

}

return false;

}

private void PrintSpecificObjectsByObjectName(string? objname)

{

Organization[] arr = organizations.ToArray();

Array.Reverse(arr);

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

{

if (arr[i].ToString() == objname)

{

Console.WriteLine(" [{0}] object name: {1}, full info: {2}", i, arr[i].ToString(), arr[i].GetFullInfo());

}

}

}

private int CountObjectsByObjectName(string? objname)

{

int count = 0;

Organization[] arr = organizations.ToArray();

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

{

if (arr[i].ToString() == objname)

{

count++;

}

}

return count;

}

private static Stack<\_Ty> RemoveFromStack<\_Ty>(Stack<\_Ty> previous\_stack, int elem\_index)

{

Stack<\_Ty> newStack = new Stack<\_Ty>();

\_Ty[] arr = previous\_stack.ToArray();

Array.Reverse(arr);

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

if (i != elem\_index)

newStack.Push(arr[i]);

newStack.Reverse();

return newStack;

}

private void DeleteOrg()

{

int idx;

do

{

idx = (int)ReadLineWithDescription(FORMAT\_TYPE.FORMAT\_INT32, "type a index of object in collection");

try

{

organizations = RemoveFromStack<Organization>(organizations, idx);

Console.WriteLine("OK Element on index {0} deleted.", idx);

PrintObjectsInCollection();

return;

}

catch (ArgumentOutOfRangeException)

{

Console.WriteLine("Out of range index. Retype please");

}

} while (true);

}

private void PrintSpecificObjects()

{

string objname = (string)ReadLineWithDescription(FORMAT\_TYPE.FORMAT\_STRING, "type object name");

Console.WriteLine("Printing objects with name {0}:", objname);

PrintSpecificObjectsByObjectName(objname);

}

private void CountSpecificObjects()

{

string objname = (string)ReadLineWithDescription(FORMAT\_TYPE.FORMAT\_STRING, "type object name");

Console.WriteLine(" {0} objects with name {1}:", CountObjectsByObjectName(objname), objname);

}

static private void PrintForeach(Stack<Organization> orgsList)

{

Organization[] arr = orgsList.ToArray();

Array.Reverse(arr);

foreach (Organization org in arr)

Console.WriteLine("object name: {0}, full info: {1}", org.ToString(), org.GetFullInfo());

Console.Write("\n");

}

private void CloneAndSortCollection()

{

Console.WriteLine("Source elements:");

PrintForeach(organizations); //print original

Stack<Organization> sortedСollectionCopy = SortStack<Organization>(new Stack<Organization>(organizations));

Console.WriteLine("Sorted elements copy:");

PrintForeach(sortedСollectionCopy); //print sorted copy

}

private void FindCompanyInCollectionByName()

{

int foundCount = 0;

string company = (string)ReadLineWithDescription(FORMAT\_TYPE.FORMAT\_STRING, "type company name for search in collection");

foreach (Organization org in organizations)

{

if (company.CompareTo(org.GetOrgName()) == 0)

{

Console.WriteLine(org.GetFullInfo());

foundCount++;

}

}

if (foundCount == 0)

Console.WriteLine("Not found result for this search\n");

}

private void BeginMenu()

{

int menuItem;

while (true)

{

menuItem = (int)ReadLineWithDescription(FORMAT\_TYPE.FORMAT\_INT32, ">>");

switch (menuItem)

{

case 0:

Console.WriteLine("performing exiting");

return;

case 1:

PrintMenu();

break;

case 2:

InsertNewOrgs();

break;

case 3:

DeleteOrg();

break;

case 4:

PrintObjectsInCollection();

break;

case 5:

PrintSpecificObjects();

break;

case 6:

CountSpecificObjects();

break;

case 7:

PrintForeach(organizations);

break;

case 8:

CloneAndSortCollection();

break;

case 9:

FindCompanyInCollectionByName();

break;

}

}

}

public void Start()

{

PrintOrganizationObjects();

InsertDefaultOrgs();

PrintMenu();

BeginMenu();

}

}

}

**CollectionsLab2.cs**

namespace lab11k

{

public class CollectionsLab2 : ConInFormat

{

private List<Organization> organizations;

public CollectionsLab2()

{

organizations = new List<Organization>();

}

private enum ORG\_TYPE

{

ORGANIZATION = 0,

FACTORY,

INSURANCE\_COMPANY,

SHIPBUILDING\_COMPANY,

LIBRARY

};

protected void InsertDefaultOrgs()

{

organizations.Add(new Organization("Организация1", "ул. Петрова 44/2", "10:00-22:00"));

organizations.Add(new Factory("Системы отопления", "ул. Аносова 2", "09:00-20:00", 12000, 10000));

organizations.Add(new InsuranceCompany("Страховое АО ВСК", "ул. 25 Октября", "09:00–18:00", 254003));

organizations.Add(new ShipbuildingCompany("Hanse LLC", "10600 1349/127 Charoennakorn Road", "10AM-10PM", 18000, 6000, 24000, 2400));

organizations.Add(new Library("Библиотека", "ул. Смирнова", "12:00-18:00", 1400));

}

void PrintOrganizationObjects()

{

object[] objects = GetObjects();

Console.WriteLine("List of objects:");

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

Console.WriteLine(" {0}. '{1}' object", i, objects[i].ToString());

}

Organization CreateOrganization(ORG\_TYPE type)

{

Organization orgobj;

switch (type)

{

default:

case ORG\_TYPE.ORGANIZATION:

orgobj = new Organization();

break;

case ORG\_TYPE.FACTORY:

orgobj = new Factory();

break;

case ORG\_TYPE.INSURANCE\_COMPANY:

orgobj = new InsuranceCompany();

break;

case ORG\_TYPE.SHIPBUILDING\_COMPANY:

orgobj = new ShipbuildingCompany();

break;

case ORG\_TYPE.LIBRARY:

orgobj = new Library();

break;

}

orgobj.FillInfo();

return orgobj;

}

private void PrintMenu()

{

Console.WriteLine(

"0. Exit\n" +

"1. Print menu\n" +

"2. Add objects to collection\n" +

"3. Remove object from collection\n" +

"4. List objects in collection\n" +

"5. Count specific objects\n" +

"6. Print specific objects\n" +

"7. Foreach print\n" +

"8. Copy and sort\n" +

"9. Find company in collection\n\n"

);

}

private void PrintObjectsInCollection()

{

Console.WriteLine("List of objects in collection:");

Organization? org;

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

{

org = organizations[i];

Console.WriteLine(" [{0}] object name: {1}, full info: {2}", i, org.ToString(), org.GetFullInfo());

}

}

private void InsertNewOrgs()

{

/\* add new organizations to collection \*/

int count = (int)ReadLineWithDescription(FORMAT\_TYPE.FORMAT\_INT32, "enter number of objects");

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

{

int orgtype = (int)ReadLineWithDescription(FORMAT\_TYPE.FORMAT\_INT32, "enter object index");

Organization org = CreateOrganization((ORG\_TYPE)orgtype);

organizations.Add(org);

Console.WriteLine("Element added on index {0}", organizations.Count - 1);

}

}

private object[] GetObjects()

{

object[] objects = {

new Organization(),

new Factory(),

new InsuranceCompany(),

new ShipbuildingCompany(),

new Library()

};

return objects;

}

private bool FindObjectByName(ref object dst, string? name)

{

object[] objects = GetObjects();

foreach (object obj in objects)

{

if (name == obj.ToString())

{

dst = obj;

return true;

}

}

return false;

}

private void PrintSpecificObjectsByObjectName(string? objname)

{

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

{

Organization? obj = organizations[i];

if (obj.ToString() == objname)

{

Console.WriteLine(" [{0}] object name: {1}, full info: {2}", i, obj.ToString(), obj.GetFullInfo());

}

}

}

private int CountObjectsByObjectName(string? objname)

{

int count = 0;

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

{

if (organizations[i].ToString() == objname)

{

count++;

}

}

return count;

}

private void DeleteOrg()

{

int idx;

do

{

idx = (int)ReadLineWithDescription(FORMAT\_TYPE.FORMAT\_INT32, "type a index of object in collection");

try

{

organizations.RemoveAt(idx);

Console.WriteLine("OK Element on index {0} deleted.", idx);

PrintObjectsInCollection();

return;

}

catch (ArgumentOutOfRangeException)

{

Console.WriteLine("Out of range index. Retype please");

}

} while (true);

}

private void PrintSpecificObjects()

{

string objname = (string)ReadLineWithDescription(FORMAT\_TYPE.FORMAT\_STRING, "type object name");

Console.WriteLine("Printing objects with name {0}:", objname);

PrintSpecificObjectsByObjectName(objname);

}

private void CountSpecificObjects()

{

string objname = (string)ReadLineWithDescription(FORMAT\_TYPE.FORMAT\_STRING, "type object name");

Console.WriteLine(" {0} objects with name {1}:", CountObjectsByObjectName(objname), objname);

}

static private void PrintForeach(List<Organization> orgsList)

{

foreach (Organization org in orgsList)

Console.WriteLine("object name: {0}, full info: {1}", org.ToString(), org.GetFullInfo());

Console.Write("\n");

}

private void CloneAndSortCollection()

{

Console.WriteLine("Source elements:");

PrintForeach(organizations); //print original

List<Organization> collectionCopy = new List<Organization>(organizations);

collectionCopy.Sort();

Console.WriteLine("Sorted elements copy:");

PrintForeach(collectionCopy); //print sorted copy

}

private void FindCompanyInCollectionByName()

{

int foundCount = 0;

string company = (string)ReadLineWithDescription(FORMAT\_TYPE.FORMAT\_STRING, "type company name for search in collection");

foreach (Organization org in organizations)

{

if (company.CompareTo(org.GetOrgName()) == 0)

{

Console.WriteLine(org.GetFullInfo());

foundCount++;

}

}

if (foundCount == 0)

Console.WriteLine("Not found result for this search\n");

}

private void BeginMenu()

{

int menuItem;

while (true)

{

menuItem = (int)ReadLineWithDescription(FORMAT\_TYPE.FORMAT\_INT32, ">>");

switch (menuItem)

{

case 0:

Console.WriteLine("performing exiting");

return;

case 1:

PrintMenu();

break;

case 2:

InsertNewOrgs();

break;

case 3:

DeleteOrg();

break;

case 4:

PrintObjectsInCollection();

break;

case 5:

PrintSpecificObjects();

break;

case 6:

CountSpecificObjects();

break;

case 7:

PrintForeach(organizations);

break;

case 8:

CloneAndSortCollection();

break;

case 9:

FindCompanyInCollectionByName();

break;

}

}

}

public void Start()

{

PrintOrganizationObjects();

InsertDefaultOrgs();

PrintMenu();

BeginMenu();

}

}

}

**TestCollections.cs**

namespace lab11k

{

public class TestCollections : ConInFormat

{

LinkedList<Organization> collection1;

Dictionary<string, Factory> collection2;

/\* get linked list node by index \*/

public static bool LinkedListGetNodeByIndex<\_Ty>(ref LinkedListNode<\_Ty>? dest, LinkedList<\_Ty> ?list, int index)

{

int i;

LinkedListNode<\_Ty>? node = null;

int hfalfCount = list.Count >> 1;

/\* check \*/

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

return false;

if(index <= hfalfCount) {

// if elem index less or equal half size, find from start

node = list.First;

for (i = 0; i < index; i++) {

if (node == null)

return false;

node = node.Next;

}

}

else

{

// find from end if elem index greater half size

node = list.Last;

for (i = list.Count - 1; i > index; i--)

{

if (node == null)

return false;

node = node.Previous;

}

}

dest = node;

return true;

}

public static bool LinkedListGetNodeDataByIndex<\_Ty>(ref \_Ty dest, LinkedList<\_Ty>? list, int index)

{

LinkedListNode<\_Ty>? node = null;

if (!LinkedListGetNodeByIndex<\_Ty>(ref node, list, index))

return false;

dest = node.Value;

return true;

}

private void CollectionsAddElement(int uniqueid, string? name, string? street, string? worktime, int planmax, int plancur)

{

Organization organization = new Organization(name, street, worktime); // add new base class object

Factory factory = new Factory(name, street, worktime, planmax, plancur); // add new derived class object

collection1.AddLast(organization);

collection2.Add($"{factory.ToString()}\_{uniqueid}", factory);

}

private bool CollectionsRemoveElement(int elemidx)

{

/\* delete node from linked list \*/

LinkedListNode<Organization>? nodeToRemove = null;

if (!LinkedListGetNodeByIndex<Organization>(ref nodeToRemove, collection1, elemidx))

{

Console.WriteLine("Failed to remove node with index {0} from {1}", elemidx, collection1.ToString());

return false; // index out of bounds

}

collection1.Remove(nodeToRemove); //remove node

/\* remove from list \*/

Factory empty = new Factory(); //create new object for get object name

string? keystr = $"{empty.ToString()}\_{elemidx}";

Console.WriteLine("Delete element by key {0}", keystr);

if (!collection2.Remove(keystr)) {

Console.WriteLine("Failed to remove key-value with index {0} from {1}", elemidx, collection2.ToString());

return false; // index out of bounds

}

return true; //OK

}

public TestCollections(int numberOfElements)

{

Random random = new Random();

collection1 = new LinkedList<Organization>();

collection2 = new Dictionary<string, Factory>();

for (int i = 0; i < numberOfElements; i++) {

int hour = random.Next(9, 13);

int mins = random.Next(0, 50);

int currhour = hour % 12;

int minval = Math.Min(hour + 1, currhour);

int maxval = Math.Max(hour + 1, currhour);

int endhour = random.Next(minval, maxval);

int endmins = random.Next(0, 50);

int planmax = random.Next(10, 1000);

int plancur = random.Next(0, planmax);

string name = $"organization{i}";

string street = $"street {i + 20}";

string worktime = $"{hour}:{mins}-{endhour}:{endmins}";

CollectionsAddElement(i, name, street, worktime, planmax, plancur);

}

}

private void PrintCollections()

{

/\* enum collection keys \*/

foreach (string? key in collection2.Keys)

Console.WriteLine("key '{0}'", key);

}

public void FindTests()

{

int elemidx;

Organization center = new Organization();

Organization first = collection1.First();

Organization end = collection1.Last();

Organization noexists = new Organization("This Organization Is No Exists!", "Ababsdbsdb", "00:00-00:00");

/\* COLLECTION 1 \*/

Console.WriteLine("COLLECTION 1");

if (!LinkedListGetNodeDataByIndex<Organization>(ref center, collection1, collection1.Count >> 1)) {

Console.WriteLine("Index out of bounds!");

return;

}

Stopwatch stopwatch = new Stopwatch();

stopwatch.Start();

collection1.Contains(first);

stopwatch.Stop();

Console.WriteLine("1. find first element in collection1. elapsed time {0} ms", stopwatch.Elapsed);

stopwatch.Reset();

stopwatch.Start();

collection1.Contains(center);

stopwatch.Stop();

Console.WriteLine("2. find center element in collection1. elapsed time {0} ms", stopwatch.Elapsed);

stopwatch.Reset();

stopwatch.Start();

collection1.Contains(end);

stopwatch.Stop();

Console.WriteLine("3. find end element in collection1. elapsed time {0} ms", stopwatch.Elapsed);

stopwatch.Reset();

stopwatch.Start();

collection1.Contains(noexists);

stopwatch.Stop();

Console.WriteLine("4. find no exists element in collection1. elapsed time {0} ms", stopwatch.Elapsed);

stopwatch.Reset();

Console.Write("\n\n");

/\* COLLECTION 2 \*/

Console.WriteLine("COLLECTION 2");

string keystr;

Factory empty = new Factory(); //create new object for get object name

keystr = $"{empty.ToString()}\_0";

stopwatch.Start();

collection2.ContainsKey(keystr);

stopwatch.Stop();

Console.WriteLine("1. find first element in collection2. elapsed time {0} ms", stopwatch.Elapsed);

stopwatch.Reset();

int centeridx = collection2.Count >> 1;

keystr = $"{empty.ToString()}\_{centeridx}";

stopwatch.Start();

collection2.ContainsKey(keystr);

stopwatch.Stop();

Console.WriteLine("2. find center element in collection2. elapsed time {0} ms", stopwatch.Elapsed);

stopwatch.Reset();

keystr = $"{empty.ToString()}\_{collection2.Count-1}";

stopwatch.Start();

collection2.ContainsKey(keystr);

stopwatch.Stop();

Console.WriteLine("3. find end element in collection2. elapsed time {0} ms", stopwatch.Elapsed);

stopwatch.Reset();

keystr = $"THIS IS NOT EXISTS KEY";

stopwatch.Start();

collection2.ContainsKey(keystr);

stopwatch.Stop();

Console.WriteLine("4. find no exists element in collection2. elapsed time {0} ms", stopwatch.Elapsed);

stopwatch.Reset();

Console.Write("\n\n");

while (true) {

PrintCollections();

elemidx = (int)ReadLineWithDescription(FORMAT\_TYPE.FORMAT\_INT32, "Type element index from delete from collections");

if(!CollectionsRemoveElement(elemidx)) {

Console.WriteLine("Failed to delete element with index {0} from collections", elemidx);

continue;

}

Console.WriteLine("Deleted element with index {0} from collections", elemidx);

}

}

}

}

**Program.cs**

namespace lab11k

{

internal class Program

{

static void Main(string[] args)

{

new CollectionsLab1().Start(); //Stack<T> 1

new CollectionsLab2().Start(); //List<T> 2

new TestCollections(5).FindTests(); // 3

}

}

}

**ConInFormat.cs**

namespace lab11k

{

public class ConInFormat

{

public enum FORMAT\_TYPE

{

FORMAT\_STRING = 0,

FORMAT\_INT16,

FORMAT\_INT32,

FORMAT\_INT64,

FORMAT\_FLOAT,

FORMAT\_DOUBLE

};

static protected object ReadLineWithDescription(FORMAT\_TYPE format\_type, string? description)

{

string ret = "empty";

string? strref = null;

do {

Console.Write("{0}: ", description);

strref = Console.ReadLine();

if (strref == null) {

Console.WriteLine("Reading from input stream failed");

return ret;

}

ret = strref;

try {

switch (format\_type) {

case FORMAT\_TYPE.FORMAT\_INT16:

return Convert.ToInt16(ret);

case FORMAT\_TYPE.FORMAT\_INT32:

return Convert.ToInt32(ret);

case FORMAT\_TYPE.FORMAT\_INT64:

return Convert.ToInt64(ret);

case FORMAT\_TYPE.FORMAT\_FLOAT:

return Convert.ToSingle(ret);

case FORMAT\_TYPE.FORMAT\_DOUBLE:

return Convert.ToDouble(ret);

case FORMAT\_TYPE.FORMAT\_STRING:

default:

return Convert.ToString(ret); //default convert to string

}

}

catch(FormatException) {

Console.WriteLine("Exception: format exception occurred. Retype data!");

continue;

}

} while (true);

}

}

}

**Factory.cs**

namespace lab11k

{

public class Factory : Organization

{

protected int done\_plan;

protected int total\_plan;

public Factory() : base("", "", "") {

done\_plan = 0;

total\_plan = 0;

subobject = new Organization("", "", "");

}

public Factory(string name, string address, string worktime, int plan, int done) : base(name, address, worktime)

{

total\_plan = plan;

done\_plan = done;

subobject = new Organization(name, address, worktime);

}

public int getPlan() { return total\_plan; }

public void setPlan(int plan) { total\_plan = plan; }

public int getDonePlan() { return done\_plan; }

public void setDonePlan(int plan) { done\_plan = plan; }

public new string GetFullInfo()

{

return $"Factory: {name}, {address}, {worktime}. (Done|Total) All({done\_plan}|{total\_plan})";

}

public override void FillInfo()

{

base.FillInfo();

done\_plan = (int)ReadLineWithDescription(FORMAT\_TYPE.FORMAT\_INT32, "Enter done plan");

total\_plan = (int)ReadLineWithDescription(FORMAT\_TYPE.FORMAT\_INT32, "Enter total plan");

}

}

}

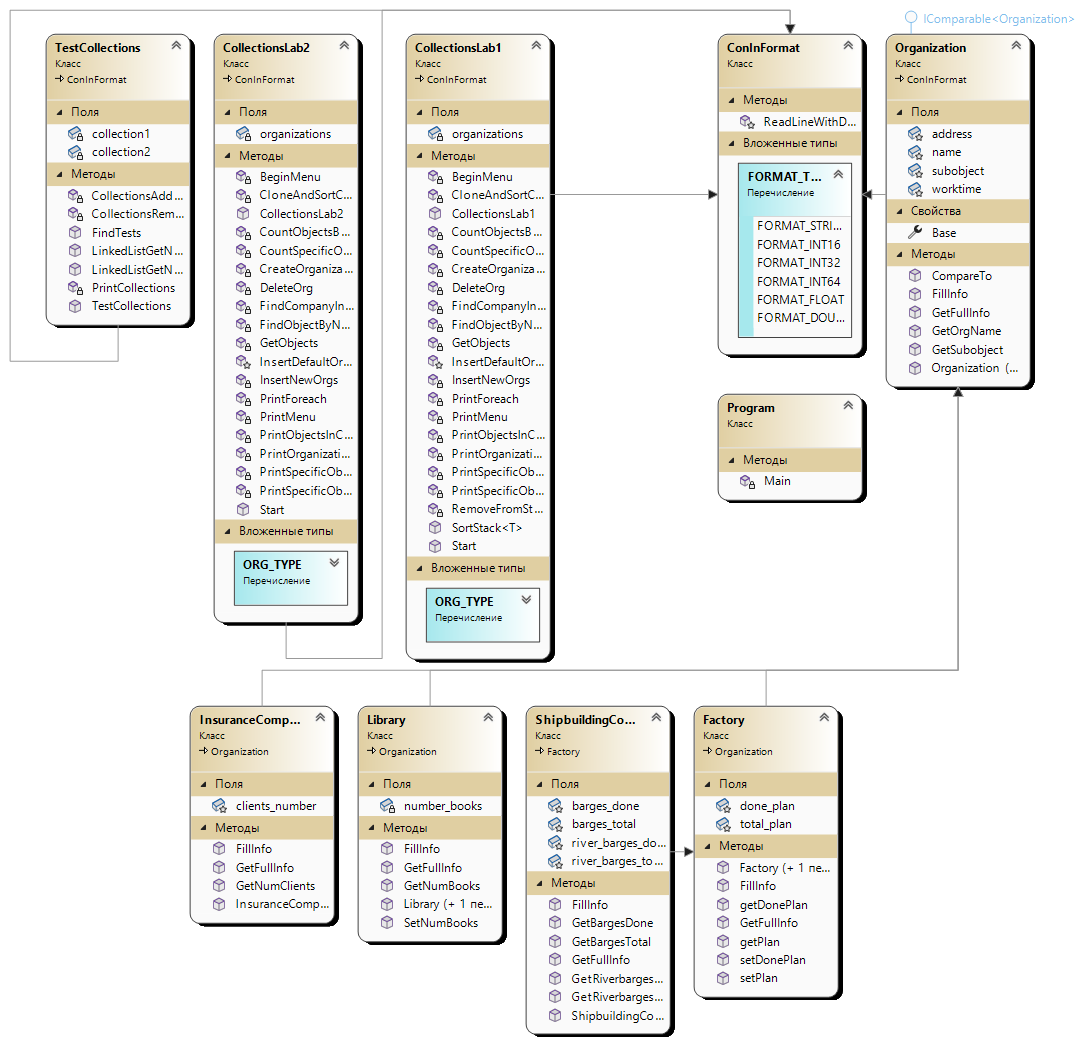


Рисунок - Диаграмма классов

**Анализ полученных результатов**

Коллекция 1

* Поиск первого элемента в коллекции занял 862мс, что говорит о том, что поиск начинается с конца, т.к это наибольшее время из всех тестов.
* Поиск центрального элемента занимает 56мс, что уже намного меньше по времени, чем поиск первого элемента.
* Поиск последнего элемента занял наименьшее время (16 мс).
* Поиск несуществующего элемента в коллекции занял 19мс.

Коллекция 2

* Поиск первого элемента в коллекции занял так же 212мс.
* Поиск центрального элемента занял всего 9мс
* Поиск последнего элемента занял 10мс
* А поиск несуществующего, занял 12 мс.

Исходя из проведенных тестирований, можно сказать, что коллекция List работает медленнее коллекции Dictionary.