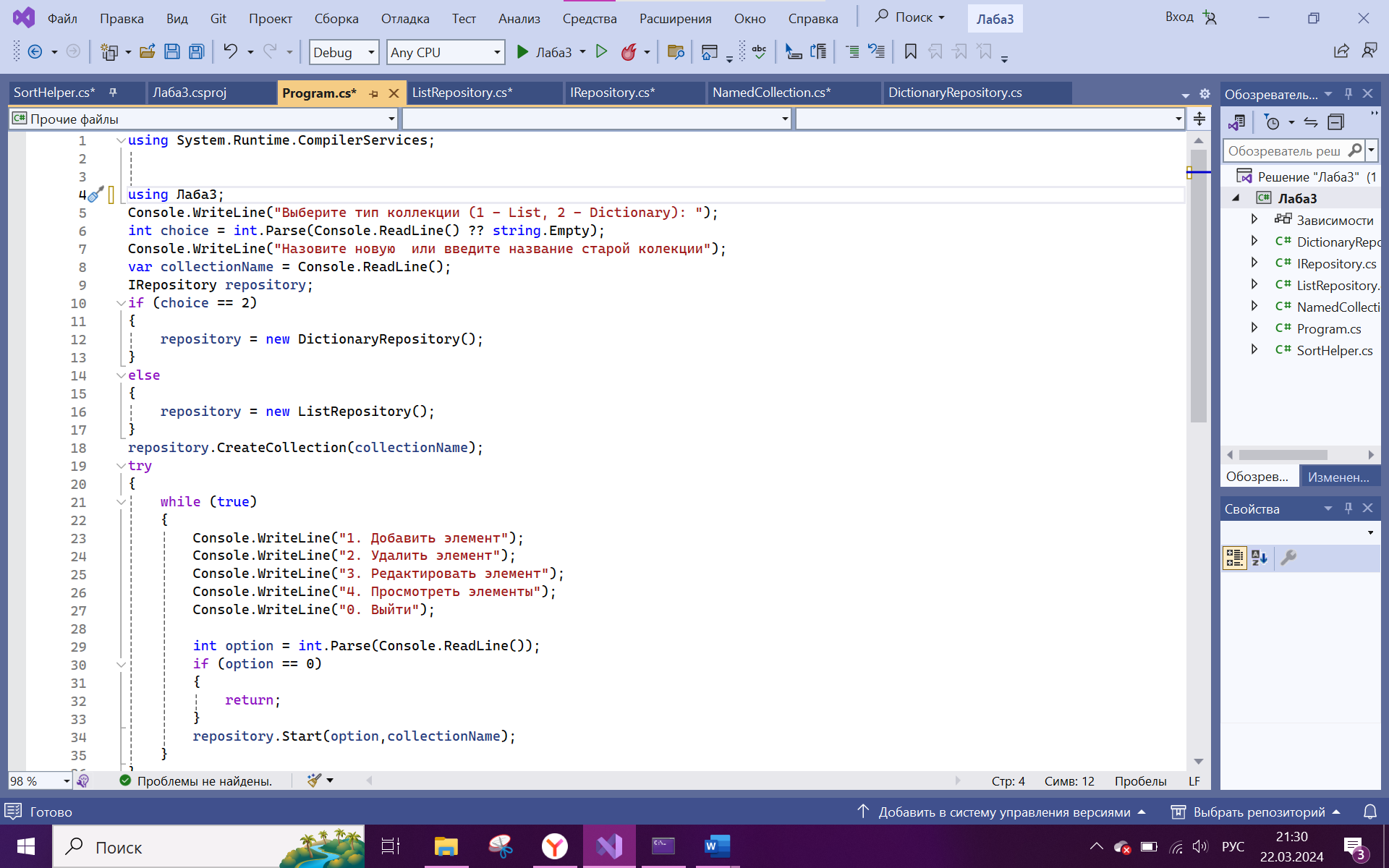
**Лабораторная работа № 3**: Работа с коллекциями в C#

Выполнила студентка группы ИСП-223 Губарева Полина

**Цель**:

Научиться работать с различными типами коллекций в C# и понимать их преимущества и недостатки.



using System.Runtime.CompilerServices;

using Лаба3;

Console.WriteLine("Выберите тип коллекции (1 - List, 2 - Dictionary): ");

int choice = int.Parse(Console.ReadLine() ?? string.Empty);

Console.WriteLine("Назовите новую или введите название старой колекции");

var collectionName = Console.ReadLine();

IRepository repository;

if (choice == 2)

{

repository = new DictionaryRepository();

}

else

{

repository = new ListRepository();

}

repository.CreateCollection(collectionName);

try

{

while (true)

{

Console.WriteLine("1. Добавить элемент");

Console.WriteLine("2. Удалить элемент");

Console.WriteLine("3. Редактировать элемент");

Console.WriteLine("4. Просмотреть элементы");

Console.WriteLine("0. Выйти");

int option = int.Parse(Console.ReadLine());

if (option == 0)

{

return;

}

repository.Start(option,collectionName);

}

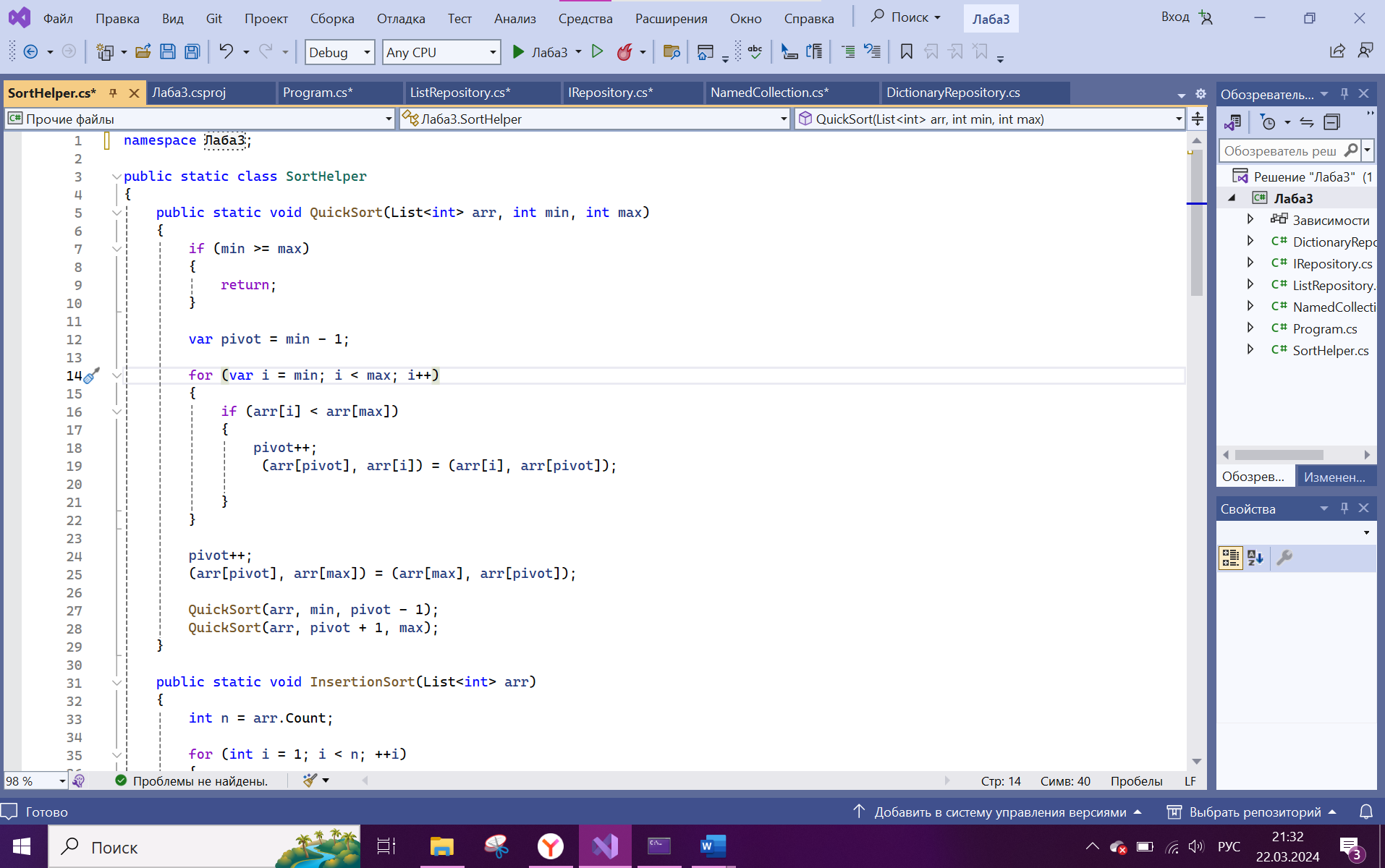
}

finally

{

repository.Save();

}



namespace Лаба3;

public static class SortHelper

{

public static void QuickSort(List<int> arr, int min, int max)

{

if (min >= max)

{

return;

}

var pivot = min - 1;

for (var i = min; i < max; i++)

{

if (arr[i] < arr[max])

{

pivot++;

(arr[pivot], arr[i]) = (arr[i], arr[pivot]);

}

}

pivot++;

(arr[pivot], arr[max]) = (arr[max], arr[pivot]);

QuickSort(arr, min, pivot - 1);

QuickSort(arr, pivot + 1, max);

}

public static void InsertionSort(List<int> arr)

{

int n = arr.Count;

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

{

int key = arr[i];

int j = i - 1;

while (j >= 0 && arr[j] > key)

{

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

public static void MergeSort(List<int> arr, int left, int right)

{

if (left < right)

{

int mid = (left + right) / 2;

MergeSort(arr, left, mid);

MergeSort(arr, mid + 1, right);

Merge(arr, left, mid, right);

}

}

static void Merge(List<int> arr, int left, int mid, int right)

{

int n1 = mid - left + 1;

int n2 = right - mid;

int[] leftArray = new int[n1];

int[] rightArray = new int[n2];

int i,j;

for (i = 0; i < n1; ++i)

leftArray[i] = arr[left + i];

for (j = 0; j < n2; ++j)

rightArray[j] = arr[mid + 1 + j];

i = 0;

j = 0;

int k = left;

while (i < n1 && j < n2)

{

if (leftArray[i] <= rightArray[j])

{

arr[k] = leftArray[i];

i++;

}

else

{

arr[k] = rightArray[j];

j++;

}

k++;

}

while (i < n1)

{

arr[k] = leftArray[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = rightArray[j];

j++;

k++;

}

}

}



<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<OutputType>Exe</OutputType>

<TargetFramework>net8.0</TargetFramework>

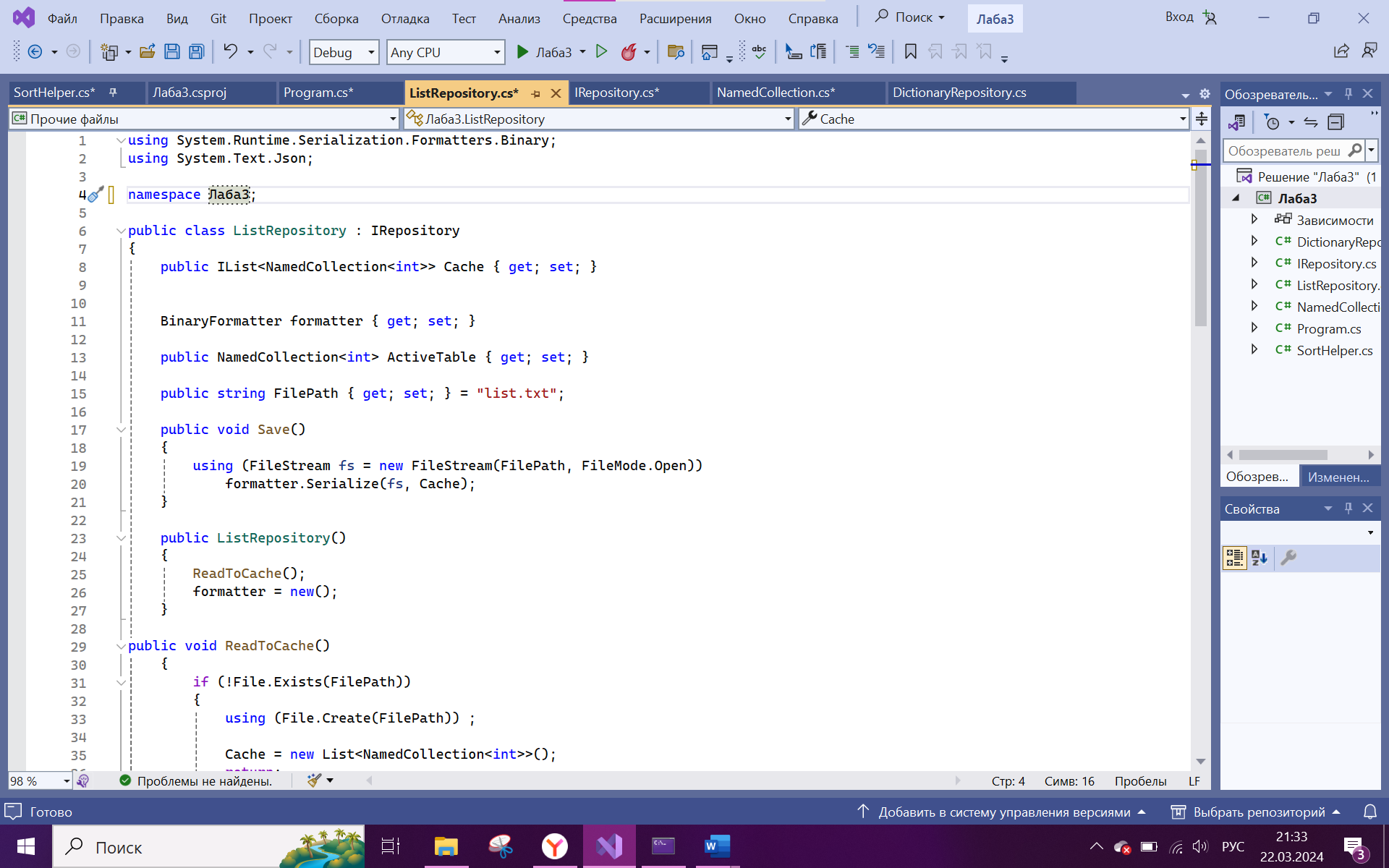
<ImplicitUsings>enable</ImplicitUsings>

<Nullable>enable</Nullable>

<EnableUnsafeBinaryFormatterSerialization>true</EnableUnsafeBinaryFormatterSerialization>

</PropertyGroup>

</Project>



using System.Runtime.Serialization.Formatters.Binary;

using System.Text.Json;

namespace Лаба3;

public class ListRepository : IRepository

{

public IList<NamedCollection<int>> Cache { get; set; }

BinaryFormatter formatter { get; set; }

public NamedCollection<int> ActiveTable { get; set; }

public string FilePath { get; set; } = "list.txt";

public void Save()

{

using (FileStream fs = new FileStream(FilePath, FileMode.Open))

formatter.Serialize(fs, Cache);

}

public ListRepository()

{

ReadToCache();

formatter = new();

}

public void ReadToCache()

{

if (!File.Exists(FilePath))

{

using (File.Create(FilePath)) ;

Cache = new List<NamedCollection<int>>();

return;

}

Cache = (IList<NamedCollection<int>>)formatter.Deserialize(File.Open(FilePath,FileMode.Open));

}

public void Start(int option,string tableName)

{

ActiveTable = Cache.FirstOrDefault(p => p.Name == tableName) ?? throw new InvalidOperationException();

if (option == 4)

{

SortHelper.QuickSort(ActiveTable.Value, 0, ActiveTable.Value.Count - 1);

ActiveTable.ViewElements();

return;

}

int n = int.Parse(Console.ReadLine());

switch (option)

{

case 1:

ActiveTable?.Add(n);

break;

case 2:

ActiveTable?.Remove(n);

break;

case 3:

Console.WriteLine("индекс");

int j = int.Parse(Console.ReadLine());

ActiveTable.Edit(j,n);;

break;

default:

Console.WriteLine("Некорректный ввод");

break;

}

}

public void CreateCollection(string tableName)

{

if (Cache.FirstOrDefault(p => p.Name == tableName) == null)

{

Cache.Add(new NamedCollection<int>()

{

Name = tableName,

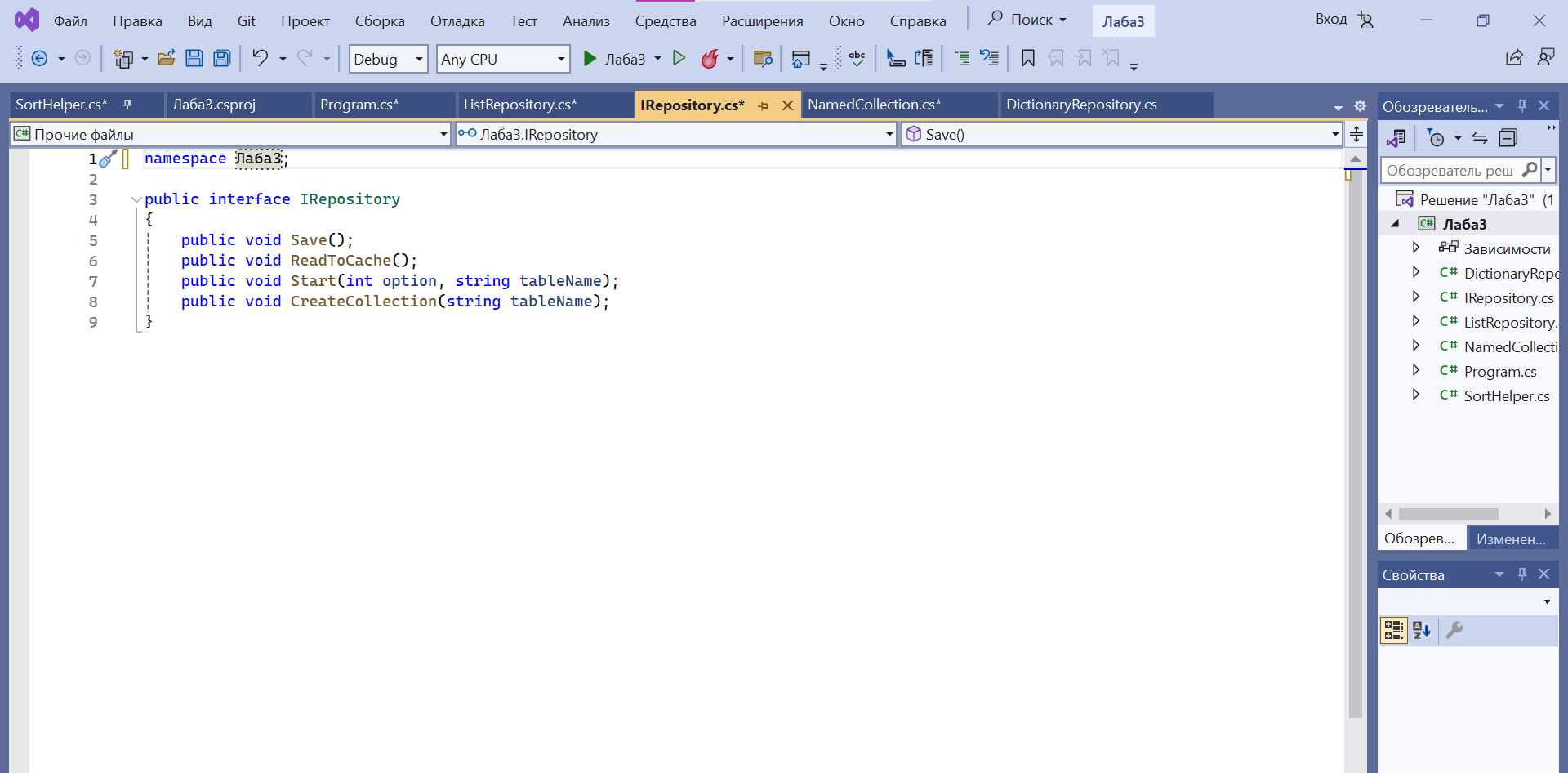
Value = new List<int>()

});

}

}

}



namespace Лаба3;

public interface IRepository

{

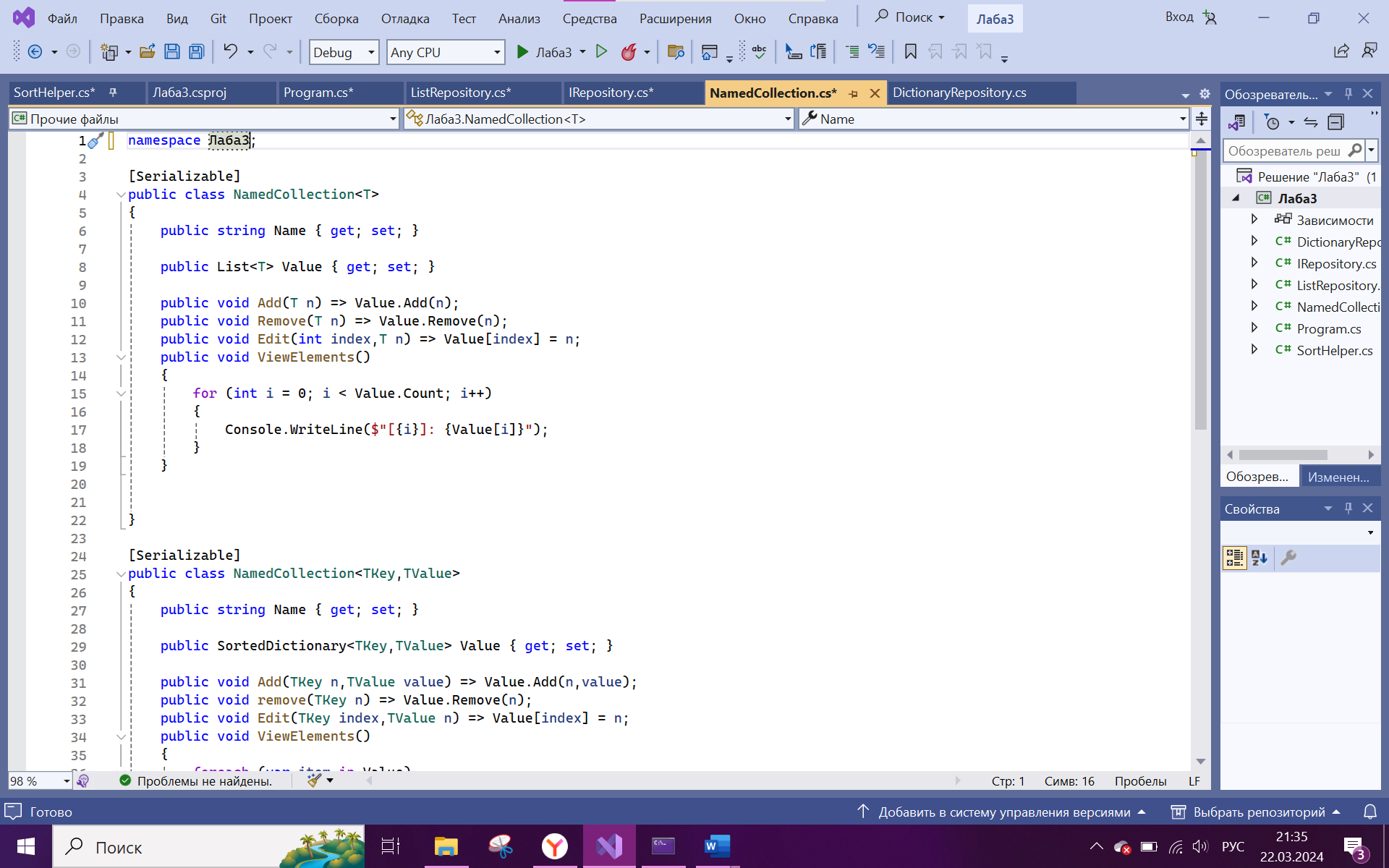
public void Save();

public void ReadToCache();

public void Start(int option, string tableName);

public void CreateCollection(string tableName);

}



namespace Лаба3;

[Serializable]

public class NamedCollection<T>

{

public string Name { get; set; }

public List<T> Value { get; set; }

public void Add(T n) => Value.Add(n);

public void Remove(T n) => Value.Remove(n);

public void Edit(int index,T n) => Value[index] = n;

public void ViewElements()

{

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

{

Console.WriteLine($"[{i}]: {Value[i]}");

}

}

}

[Serializable]

public class NamedCollection<TKey,TValue>

{

public string Name { get; set; }

public SortedDictionary<TKey,TValue> Value { get; set; }

public void Add(TKey n,TValue value) => Value.Add(n,value);

public void remove(TKey n) => Value.Remove(n);

public void Edit(TKey index,TValue n) => Value[index] = n;

public void ViewElements()

{

foreach (var item in Value)

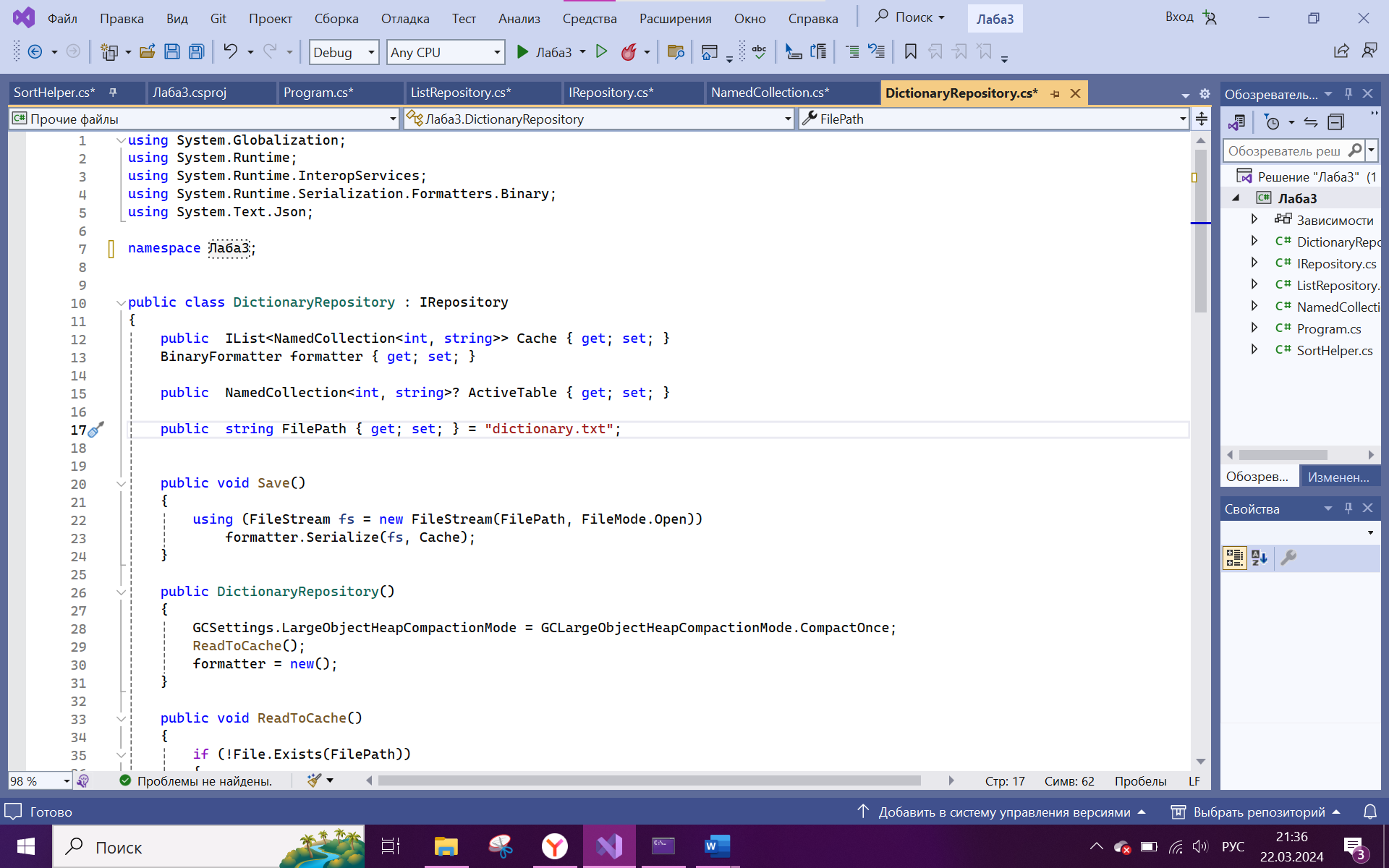
{

Console.WriteLine($"[{item.Key}]: {item.Value}");

}

}

}



using System.Globalization;

using System.Runtime;

using System.Runtime.InteropServices;

using System.Runtime.Serialization.Formatters.Binary;

using System.Text.Json;

namespace Лаба3;

public class DictionaryRepository : IRepository

{

public IList<NamedCollection<int, string>> Cache { get; set; }

BinaryFormatter formatter { get; set; }

public NamedCollection<int, string>? ActiveTable { get; set; }

public string FilePath { get; set; } = "dictionary.txt";

public void Save()

{

using (FileStream fs = new FileStream(FilePath, FileMode.Open))

formatter.Serialize(fs, Cache);

}

public DictionaryRepository()

{

GCSettings.LargeObjectHeapCompactionMode = GCLargeObjectHeapCompactionMode.CompactOnce;

ReadToCache();

formatter = new();

}

public void ReadToCache()

{

if (!File.Exists(FilePath))

{

using (File.Create(FilePath)) ;

Cache = new List<NamedCollection<int,string>>();

return;

}

Cache = (IList<NamedCollection<int,string>>)formatter.Deserialize(File.Open(FilePath,FileMode.Open));

}

public void Start(int option,string tableName)

{

if(ActiveTable == null)

ActiveTable = Cache.FirstOrDefault(p => p.Name == tableName);

if (option == 4)

{

ActiveTable.ViewElements();

return;

}

string j = null;

if (option is 1

or 3)

{

Console.WriteLine("значение");

j = Console.ReadLine();

}

int n = int.Parse(Console.ReadLine());

switch (option)

{

case 1:

ActiveTable?.Add(n,j);

break;

case 2:

ActiveTable?.Value.Remove(n);

break;

case 3:

ActiveTable.Edit(n,j);

break;

default:

Console.WriteLine("Некорректный ввод");

break;

}

}

public void CreateCollection(string tableName)

{

if (Cache.FirstOrDefault(p => p.Name == tableName) == null)

{

Cache.Add(new NamedCollection<int,string>()

{

Name = tableName,

Value = new SortedDictionary<int, string>()

});

}

}

}

