# Лабораторная работа № 14 LINQ to Objects

**Цель.** Получить практические навыки использования запросов LINQ to objects.

## Постановка задачи

**Часть 1**

1. Сформировать обобщенную стандартную коллекцию, содержащую ссылки на другие стандартные обобщенные коллекции.
2. Заполнить коллекции объектами иерархии классов (лабораторная работа №10).

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

Запросы должны быть выполнены двумя способами:

1. С использованием LINQ запросов.
2. С использованием методов расширения.

Каждый запрос выполняется в отдельной функции.

Примеры запросов (лабораторная работа №10).

**Часть 2**

1. В коллекцию MyNewCollection (лаб. раб. 12) добавить 3 метода расширения, реализующие следующие запросы:

1. На выборку данных по условию.
2. Агрегирование данных (среднее, максимум/минимум, сумма и пр.).
3. Сортировка коллекции (по убыванию/по возрастанию).

**Дополнительное задание:**

1. Группировка данных.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Коллекция\_1 | Коллекция\_2 | Иерархия классов |
| 4 | Город (SortedDictionary) | Район (Queue) | **организация,** страховая компания, судостроительная компания, завод, библиотека; |

**Исходный код**

**AddressEqualityComparer.cs**

namespace lab14k

{

internal class AddressEqualityComparer : IEqualityComparer<Organization>

{

public bool Equals(Organization x, Organization y)

{

// Сравниваем организации по их адресам

return x.address == y.address;

}

public int GetHashCode(Organization obj)

{

// Возвращаем хэш-код на основе адреса

return obj.address.GetHashCode();

}

}

}

**Extension.cs**

namespace lab14k

{

public static class Extension

{

public static int Count(this Queue<Organization> collection, Func<Organization, bool> predicate)

{

int count = collection.Where(predicate).Count();

return count;

}

public static string Aggregate(this Queue<Organization> queue, string separator)

{

if (queue == null)

{

throw new ArgumentNullException(nameof(queue));

}

return queue.Select(organization => organization.GetFullInfo()).Aggregate((result, fullInfo) => result + separator + fullInfo);

}

}

}

**Factory.cs**

namespace lab14k

{

public class Factory : Organization

{

protected int done\_plan;

protected int total\_plan;

public Factory() {

total\_plan = 0;

done\_plan = 0;

}

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

{

this.total\_plan = plan;

this.done\_plan = done\_plan;

}

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

public void setPlan(int plan) { this.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 new void AutoFill()

{

FillDefault("Factory");

}

public override string ToString()

{

return GetFullInfo();

}

}

}

**InsuranceCompany.cs**

namespace lab14k

{

public class InsuranceCompany : Organization

{

protected int clients\_number;

public InsuranceCompany() {

clients\_number = 0;

}

public InsuranceCompany(string name, string address, string worktime, int clients) : base(name, address, worktime)

{

clients\_number = clients;

}

public int GetNumClients() { return clients\_number; }

public new string GetFullInfo()

{

return $"InsuranceCompany: {name}, {address}, {worktime}. Clients({clients\_number})";

}

public new void AutoFill()

{

FillDefault("InsuranceCompany");

}

public override string ToString()

{

return GetFullInfo();

}

}

}

**Lab.cs**

namespace lab14k

{

using District = Queue<Organization>;

using City = SortedDictionary<string, Queue<Organization>>;

public class Lab

{

public Lab() {}

public void Start()

{

var city = new City();

var districts = new District();

districts.Enqueue(new Organization("PSTU", "adr", "12-00"));

districts.Enqueue(new Organization("Org", "adr", "12-00"));

city.Add("Perm", districts);

Work1(city);

HorizontalLine();

Work2(city);

HorizontalLine();

Work3(city);

HorizontalLine();

Work4(city);

HorizontalLine();

Work5(city);

}

public void HorizontalLine()

{

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

}

// выборка

public void Work1(City city)

{

Console.WriteLine("Select");

var organisations1 = city.First().Value.Where(c => c.name == "PSTU").Select(o => o.GetFullInfo());

var organisations2 = from c in city.First().Value

where c.name == "PSTU"

select c.GetFullInfo();

Console.WriteLine(string.Join(',', organisations1.ToArray()));

Console.WriteLine(string.Join(',', organisations2.ToArray()));

}

// Получение счетчика

public void Work2(City city)

{

Console.WriteLine("Count");

var organisations1Count = city.First().Value.Count(e => e.name == "PSTU");

var organisations2Count = (from c in city.First().Value where c.name == "PSTU" select c).Count();

Console.WriteLine(organisations1Count);

Console.WriteLine(organisations1Count);

}

// Использование операций над множествами

public void Work3(City city)

{

Console.WriteLine("Intersection");

var city2 = new City();

var districts2 = new District();

districts2.Enqueue(new Organization("PSTU-Corp2", "adr", "14-00"));

city2.Add("Perm", districts2);

var intersection1 = city.First().Value.Intersect(city2.First().Value, new AddressEqualityComparer());

Console.WriteLine("Intersection1:");

foreach (var item in intersection1)

{

Console.WriteLine(item);

}

var intersection2 = (from c in city.First().Value select c).Intersect(city2.First().Value, new AddressEqualityComparer());

Console.WriteLine("Intersection2:");

foreach (var item in intersection2)

{

Console.WriteLine(item);

}

}

// Агрегирование данных

public void Work4(City city)

{

Console.WriteLine("Aggregate");

var eurasiaStateAggregate1 = city.First().Value.Aggregate("|");

var eurasiaStateAggregate2 = string.Join('|', (from c in city.First().Value select c.GetFullInfo()).ToArray());

Console.WriteLine(eurasiaStateAggregate1);

Console.WriteLine(eurasiaStateAggregate2);

}

// Группировка данных

public void Work5(City city)

{

Console.WriteLine("Group");

var group1 = city.First().Value.GroupBy(e => e.address);

Console.WriteLine("Group1:");

foreach (var item in group1)

{

Console.WriteLine(item.Key + ":");

foreach (var orgs in item)

{

Console.WriteLine("\t" + orgs.GetFullInfo());

}

}

var group2 = from c in city.First().Value group c by c.address;

Console.WriteLine("\nGroup2:");

foreach (var item in group2)

{

Console.WriteLine(item.Key + ":");

foreach (var orgs in item)

{

Console.WriteLine("\t" + orgs.GetFullInfo());

}

}

}

}

}

**Library.cs**

namespace lab14k

{

public class Library : Organization

{

private int number\_books;

public Library() {

number\_books = 0;

}

public Library(string name, string address, string worktime, int num\_books) : base(name, address, worktime)

{

number\_books = num\_books;

}

public int GetNumBooks() { return number\_books; }

public void SetNumBooks(int n) { number\_books = n; }

public new string GetFullInfo()

{

return $"Library: {name}, {address}, {worktime}. Number of books: {number\_books}";

}

public new void AutoFill()

{

FillDefault("Library");

}

public override string ToString()

{

return GetFullInfo();

}

}

}

**Organization.cs**

namespace lab14k

{

public class Organization : IComparable<Organization>

{

public string name;

public string address;

public string worktime;

protected void FillDefault(string prefix)

{

Random rand = new Random();

name = $"{prefix}\_{rand.Next()}";

address = $"address {rand.Next(0, 2000)}";

int startHour = rand.Next(8, 12);

int startMins = rand.Next(0, 30);

int endHour = rand.Next(startHour + 1, startHour + 8) % 24;

int endMins = rand.Next(0, 30);

worktime = $"{startHour}:{startMins}-{endHour}:{endMins}";

}

public Organization()

{

name = "empty";

address = "empty";

worktime = "empty";

}

public int CompareTo(Organization? other)

{

return name.CompareTo(other.name);

}

public Organization(string \_name, string \_address, string \_worktime)

{

name = \_name;

address = \_address;

worktime = \_worktime;

}

public string GetFullInfo() { return $"{name}, {address}, {worktime} --"; }

public void AutoFill()

{

FillDefault("Organization");

}

public override string ToString()

{

return GetFullInfo();

}

}

}

**ShipbuildingCompany.cs**

namespace lab14k

{

/\* судостроительная \*/

public class ShipbuildingCompany : Factory

{

protected int barges\_total;

protected int barges\_done;

protected int river\_barges\_total;

protected int river\_barges\_done;

public ShipbuildingCompany() {

barges\_total = 0;

barges\_done = 0;

river\_barges\_total = 0;

river\_barges\_done = 0;

}

public ShipbuildingCompany(string name, string address, string worktime, int barges\_total, int barges\_done, int rbarges\_total, int rbarges\_done)

: base(name, address, worktime, barges\_total + rbarges\_total, barges\_done + rbarges\_done)

{

this.barges\_total = barges\_total;

this.barges\_done = barges\_done;

this.river\_barges\_total = rbarges\_total;

this.river\_barges\_done = rbarges\_done;

}

public int GetBargesTotal() { return barges\_total; }

public int GetBargesDone() { return barges\_done; }

public int GetRiverbargesTotal() { return river\_barges\_total; }

public int GetRiverbargesDone() { return river\_barges\_done; }

public new string GetFullInfo()

{

return $"ShipbuildingCompany: {name}, {address}, {worktime} (Total|Done) All({total\_plan}|{done\_plan}) barges({barges\_total}|{barges\_done}) riverbarges({river\_barges\_total}|{river\_barges\_done})";

}

public new void AutoFill()

{

FillDefault("ShipbuildingCompany");

}

public override string ToString()

{

return GetFullInfo();

}

}

}

**Program.cs**

namespace lab14k

{

public class Program

{

static void Main(string[] args) {

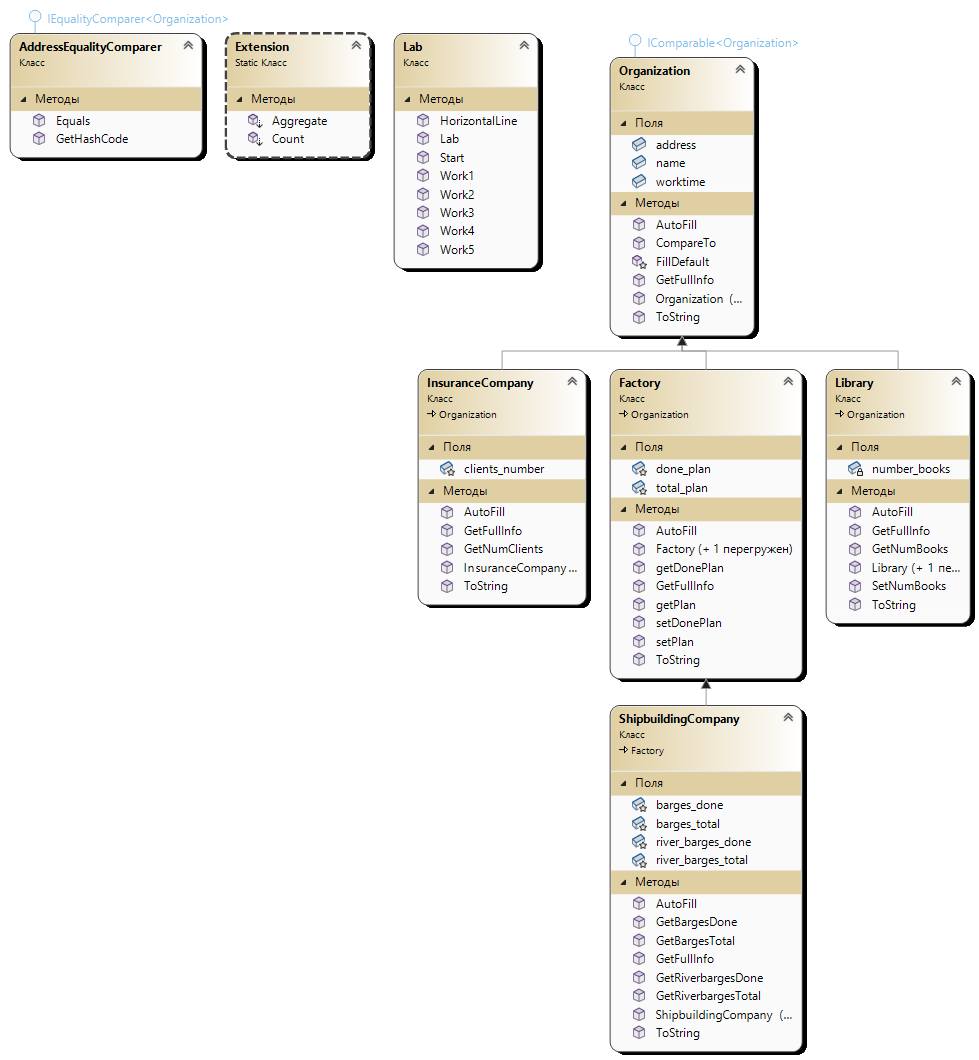
new Lab().Start();

}

}

}

**Диаграмма классов**

****