# Лабораторная работа № 14
# LINQ to Objects

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

## 1. 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):

a) На выборку данных.
b) Получение счетчика (количества объектов с заданным параметром).
c) Использование операций над множествами (пересечение, объединение, разность).
d) Агрегирование данных.
e) Группировка данных

Запросы должны быть выполнены двумя способами:

a) С использованием LINQ запросов.
b) 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:

a) На выборку данных по условию.
b) Агрегирование данных (среднее, максимум/минимум, сумма и пр.).
c) Сортировка коллекции (по убыванию/по возрастанию).

**Дополнительное задание:**

a) 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**