## Лабораторная работа №17

**Тема: Разработка программ с использованием обобщений и коллекций**.

**Цель:** Научиться разрабатывать программы с использованием обобщений и коллекций.

### Теоретические сведения:

Термин **обобщение**, по существу, означает параметризированный тип. Особая роль параметризированных типов состоит в том, что они позволяют создавать классы, структуры, интерфейсы, методы и делегаты, в которых обрабатываемые данные указываются в виде параметра. С помощью обобщений можно, например, создать единый класс, который автоматически становится пригодным для обработки разнотипных данных. Класс, структура, интерфейс, метод или делегат, оперирующий параметризированным типом данных, называется обобщенным, как, например, обобщенный класс или обобщенный метод.

Следует особо подчеркнуть, что в C# всегда имелась возможность создавать обобщенный код, оперируя ссылками типа object. А поскольку класс object является базовым для всех остальных классов, то по ссылке типа object можно обращаться к объекту любого типа. Таким образом, до появления обобщений для оперирования разнотипными объектами в программах служил обобщенный код, в котором для этой цели использовались ссылки типа object.

Но дело в том, что в таком коде трудно было соблюсти типовую безопасность, поскольку для преобразования типа object в конкретный тип данных требовалось приведение типов. А это служило потенциальным источником ошибок из-за того, что приведение типов могло быть неумышленно выполнено неверно. Это затруднение позволяют преодолеть обобщения, обеспечивая типовую безопасность, которой раньше так недоставало. Кроме того, обобщения упрощают весь процесс, поскольку исключают необходимость выполнять приведение типов для преобразования объекта или другого типа обрабатываемых данных. Таким образом, обобщения расширяют возможности повторного использования кода и позволяют делать это надежно и просто.

Обобщения — это не совсем новая конструкция; подобные концепции присутствуют и в других языках. Например, схожие с обобщениями черты имеют шаблоны C++. Однако между шаблонами C++ и обобщениями .NET есть большая разница. В C++ при создании экземпляра шаблона с конкретным типом необходим исходный код шаблонов. В отличие от

шаблонов C++, обобщения являются не только конструкцией языка C#, но также определены для CLR. Это позволяет создавать экземпляры шаблонов с определенным типом-параметром на языке Visual Basic, даже если обобщенный класс определен на C#.

В C# **коллекция** представляет собой совокупность объектов. В среде .NET Framework имеется немало интерфейсов и классов, в которых определяются и реализуются различные типы коллекций. Коллекции упрощают решение многих задач программирования благодаря тому, что предлагают готовые решения для создания целого ряда типичных, но порой трудоемких для разработки структур данных. Например, в среду .NET Framework встроены коллекции, предназначенные для поддержки динамических массивов, связных списков, стеков, очередей и хеш-таблиц. Коллекции являются современным технологическим средством, заслуживающим пристального внимания всех, кто программирует на C#.

В среде .NET Framework поддерживаются пять типов коллекций: необобщенные, специальные, с поразрядной организацией, обобщенные и параллельные.

- Необобщенные коллекции

Реализуют ряд основных структур данных, включая динамический массив, стек, очередь, а также словари, в которых можно хранить пары "ключ-значение". В отношении необобщенных коллекций важно иметь в виду следующее: они оперируют данными типа object. Таким образом, необобщенные коллекции могут служить для хранения данных любого типа, причем в одной коллекции допускается наличие разнотипных данных. Очевидно, что такие коллекции не типизированы, поскольку в них хранятся ссылки на данные типа object. Классы и интерфейсы необобщенных коллекций находятся в пространстве имен System.Collections.

- Специальные коллекции

Оперируют данными конкретного типа или же делают это каким-то особым образом. Например, имеются специальные коллекции для символьных строк, а также специальные коллекции, в которых используется однонаправленный список. Специальные коллекции объявляются в пространстве имен System.Collections.Specialized.

- Поразрядная коллекция

В прикладном интерфейсе Collections API определена одна коллекция с поразрядной организацией — это BitArray. Коллекция типа BitArray поддерживает поразрядные операции, т.е. операции над отдельными двоичными разрядами, например И, ИЛИ, исключающее ИЛИ, а следовательно, она существенно отличается своими возможностями от

остальных типов коллекций. Коллекция типа BitArray объявляется в пространстве имен System.Collections.

- Обобщенные коллекции

Обеспечивают обобщенную реализацию нескольких стандартных структур данных, включая связные списки, стеки, очереди и словари. Такие коллекции являются типизированными в силу их обобщенного характера. Это означает, что в обобщенной коллекции могут храниться только такие элементы данных, которые совместимы по типу с данной коллекцией. Благодаря этому исключается случайное несовпадение типов. Обобщенные коллекции объявляются в пространстве имен System.Collections.Generic.

- Параллельные коллекции

Поддерживают многопоточный доступ к коллекции. Это обобщенные коллекции, определенные в пространстве имен System.Collections.Concurrent.

В пространстве имен System.Collections.ObjectModel находится также ряд классов, поддерживающих создание пользователями собственных обобщенных коллекций.

Основополагающим для всех коллекций является понятие перечислителя, который поддерживается в необобщенных интерфейсах IEnumerator и IEnumerable, а также в обобщенных интерфейсах IEnumerator<T> и IEnumerable<T>. Перечислитель обеспечивает стандартный способ поочередного доступа к элементам коллекции. Следовательно, он перечисляет содержимое коллекции. В каждой коллекции должна быть реализована обобщенная или необобщенная форма интерфейса IEnumerable, поэтому элементы любого класса коллекции должны быть доступны посредством методов, определенных в интерфейсе IEnumerator или IEnumerator<T>. Это означает, что, внеся минимальные изменения в код циклического обращения к коллекции одного типа, его можно использовать для аналогичного обращения к коллекции другого типа. Любопытно, что для поочередного обращения к содержимому коллекции в цикле foreach используется перечислитель.

**Выполнение работы:**

Пример выполнения лабораторной работы.

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;

namespace Lab4
{

    class Program
    {
        static void Main()
```

```
{
    try
    {
        var st1 = new Student
        {
            Weight = 60,
            Height = 190,
            FirstName = "Marie",
            LastName = "Little",
            University = "BSTU"
        };

        var st2 = new Student
        {
            Weight = 54,
            Height = 172,
            FirstName = "Sue",
            LastName = "Jackson",
            University = "BSTU"
        };

        var st3 = new Student
        {
            Weight = 54,
            Height = 181,
            FirstName = "Lance",
            LastName = "Knight",
            University = "BSU"
        };

        var st4 = new Student
        {
            Weight = 78,
            Height = 184,
            FirstName = "Lance",
            LastName = "Stepth",
            University = "BSU"
        };

        var st5 = new Student
        {
            Weight = 81,
            Height = 184,
            FirstName = "Wesley",
            LastName = "Jackson",
            University = "BSTU"
        };

        var wr1 = new Worker
        {
            Weight = 67,
            Height = 190,
            FirstName = "Douglas",
            LastName = "Collins",
            Salary = 578.4
        };

        var wr2 = new Worker
        {
            Weight = 67,
            Height = 190,
            FirstName = "Lynn",
            LastName = "Gibson",
            Salary = 976.5
        };
```

```
                var wr3 = new Worker
                {
                    Weight = 55,
                    Height = 172,
                    FirstName = "Olivi",
                    LastName = "Smith",
                    Salary = 493
                };

                var container1 = new HumanContainer<Human> { st1, st2, wr1, wr2 };
                container1.Remove(wr2);
                container1.Remove(st1);
                //container1[-1] = st1;
                //container1[6] = st1;
                //container1[1] = st1;
                foreach (var human in container1)
                {
                    Console.WriteLine(human.ToString());
                }

                var container2 = new HumanContainer<Human>();
                container2.Add(st3);
                container2.Add(st4);
                container2.Add(st5);
                container2.Add(wr3);

                container2.Sort();

                foreach (var human in container2)
                {
                    Console.WriteLine(human.ToString());
                }

                var list = new List<HumanContainer<Human>>();
                list.Add(container1);
                list.Add(container2);

                //orderBy
                Console.WriteLine("\nLinq To objects: OrderBy, ThenBy");
                var orderRes = container1.OrderBy(h => h.Height).ThenBy(h => h.Weight);
                foreach (var human in orderRes)
                    Console.WriteLine(human);

                //where
                Console.WriteLine("\nLinq To objects: Where");
                var whereRes = container1.Where(h => (h.Height > 170 && h.Weight >= 58) ||
h.FullName.StartsWith("L"));
                foreach (var human in whereRes)
                    Console.WriteLine(human.ToString());

                //select
                Console.WriteLine("\nLinq To objects: Select");
                var selectRes = container1.Select((h, i) => new { Index = i + 1,
h.FullName });
                foreach (var el in selectRes)
                {
                    Console.WriteLine(el);
                }

                //selectMany
                Console.WriteLine("\nLinq To objects: SelectMany");
                var selectManyRes = container1.SelectMany(h => h.FullName.Split(' '));
                foreach (var el in selectManyRes)
                    Console.WriteLine(el);
```

```
                //Skip
                Console.WriteLine("\nLinq To objects: Skip");
                var skipRes = container1.Skip(2);
                foreach (var human in skipRes)
                {
                    Console.WriteLine(human);
                }

                //SkipWhile
                Console.WriteLine("\nLinq To objects: SkipWhile");
                var skipWhileRes = container1.SkipWhile(h => h.Height < 190);
                foreach (var human in skipWhileRes)
                {
                    Console.WriteLine(human);
                }

                //Take
                Console.WriteLine("\nLinq To objects: Take");
                var takeRes = container1.Take(2);
                foreach (var human in takeRes)
                {
                    Console.WriteLine(human);
                }

                //TakeWhile
                Console.WriteLine("\nLinq To objects: TakeWhile");
                var takeWhileRes = container1.TakeWhile(h => h.Height < 190);
                foreach (var human in takeWhileRes)
                {
                    Console.WriteLine(human);
                }

                //Concat
                Console.WriteLine("\nLinq To objects: Concat");
                var concatRes = container1.Concat(container2);
                foreach (var human in concatRes)
                {
                    Console.WriteLine(human);
                }

                //GroupBy
                Console.WriteLine("\nLinq To objects: GroupBy");
                var groupByRes = concatRes.Where(h => h is Student).GroupBy(h =>
((Student)h).University);
                foreach (var group in groupByRes)
                {
                    Console.WriteLine($"Group: {group.Key}, Count: {group.Count()}");
                    foreach (var human in group) Console.WriteLine(human);
                }

                //First
                Console.WriteLine("\nLinq To objects: First");
                var firstRes = concatRes.First(h => h.FullName.Length > 12);
                Console.WriteLine(firstRes);

                //FirstOrDefault
                Console.WriteLine("\nLinq To objects: FirstOrDefault");
                var firstOrDefRes = concatRes.FirstOrDefault(h => h.FullName.Length > 14);
                if (firstOrDefRes != null)
                    Console.WriteLine();

                //DefaultIfEmpty
                Console.WriteLine("\nLinq To objects: DefaultIfEmpty");
                var defaultIfEmptyRes = container2.Where(c => c.FirstName == "Eleanor")
```

```
                    .DefaultIfEmpty(new Human
                    {
                        FirstName = "Eleanor",
                        LastName = "Fuller"
                    })
                    .First();

                Console.WriteLine(defaultIfEmptyRes);

                //Min
                Console.WriteLine("\nLinq To objects: Min");
                var minRes = container1.Min(h => h.Weight);
                Console.WriteLine(minRes);

                //Max
                Console.WriteLine("\nLinq To objects: Max");
                var maxRes = container1.Max(h => h.Height);
                Console.WriteLine(maxRes);

                //Join
                Console.WriteLine("\nLinq To objects: Join");
                var joinRes = container1.Join(container2, o => o.Height, i => i.Height,
(o, i) => new Human
                {
                    FirstName = o.FirstName + " " + i.FirstName,
                    LastName = o.LastName + " " + i.LastName,
                    Height = o.Height,
                    Weight = (o.Weight + i.Weight) / 2
                });
                foreach (var human in joinRes)
                    Console.WriteLine(human);

                //GroupJoin
                Console.WriteLine("\nLinq To objects: GroupJoin");
                var groupJoinRes = container2.GroupJoin(container2, o => o.Height, i =>
i.Height, (o, i) => new
                {
                    FullName = $"{o.FirstName} {o.LastName}",
                    Count = i.Count(),
                    TotalWeight = i.Sum(s => s.Weight)
                });
                foreach (var human in groupJoinRes)
                {
                    Console.WriteLine($"{human.FullName}: Count = {human.Count},
TotalWeight: {human.TotalWeight}");
                }

                //All and Any
                Console.WriteLine("\nLinq To objects: All/Any");
                var allAnyRes = list.First(c => c.All(h => h.Height > 160) && c.Any(h => h
is Worker))
                    .Select(h => h.FirstName)
                    .OrderByDescending(s => s);

                foreach (var name in allAnyRes)
                    Console.WriteLine(name);


                //Contains
                Console.WriteLine("\nLinq To objects: Contains");
                var containsRes = list.Where(c => c.Contains(wr3))
                    .SelectMany(c => c.SelectMany(h => h.FullName.Split(' ')))
                    .Distinct()
                    .OrderBy(s => s)
                    .ToList();
```

```
                foreach (var name in containsRes)
                    Console.WriteLine(name);

            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }

    public interface IHuman
    {
        string FirstName { get; set; }
        string LastName { get; set; }
        int Height { get; set; }
        double Weight { get; set; }
    }

    public class Human : IHuman, IComparable<Human>
    {
        #region Propeties

        public string FirstName { get; set; }
        public string LastName { get; set; }
        public int Height { get; set; }
        public double Weight { get; set; }

        public string FullName
        {
            get { return string.Format("{0} {1}", FirstName, LastName); }
        }

        #endregion

        #region Methods

        public int CompareTo(Human other)
        {
            return string.Compare(other.FullName, FullName,
StringComparison.InvariantCultureIgnoreCase);
        }

        public override string ToString()
        {
            return string.Format("Class Human: \n FullName: {0}, Height: {1}, Width: {2}",
FullName,
                Height, Weight);
        }

        #endregion
    }

    public class Worker : Human
    {
        #region Properties

        public double Salary { get; set; }

        #endregion

        #region Methods

        public void DoWork() { }
```

```
        public override string ToString()
        {
            return string.Format(
                "Class Worker: \n FullName: {0}, Height: {1}, Width: {2}, Salary: {3}",
                FullName,
                Height,
                Weight,
                Salary
              );
        }

        #endregion
    }

    public class Student : Human
    {
        #region Properties

        public string University { get; set; }

        #endregion

        #region Methods

        public void DoStudy() { }

        public override string ToString()
        {
            return string.Format(
                "Class Student: \n FullName: {0}, Height: {1}, Width: {2}, University:
{3}",
                FullName,
                Height,
                Weight,
                University
             );
        }

        #endregion
    }

    public class HumanContainer<T> : IEnumerable<T> where T : Human
    {
        #region Fields

        private readonly List<T> _container;

        #endregion

        #region Constructors

        public HumanContainer()
        {
            _container = new List<T>();
        }

        #endregion

        #region Properties

        public int Count
        {
            get { return _container.Count; }
        }
```

```
        #endregion

        #region Indexers

        public T this[int index]
        {
            get
            {
                if (index < 0 || index >= Count)
                    throw new IndexOutOfRangeException();

                return _container[index];
            }
            set
            {
                if (index < 0 || index >= Count)
                    throw new IndexOutOfRangeException();

                _container[index] = value;
            }
        }

        #endregion

        #region Methods

        public T GetByName(string name)
        {
            return
                _container.FirstOrDefault(
                    h => string.Compare(h.FirstName, name,
StringComparison.InvariantCultureIgnoreCase) == 0);
        }

        public void Add(T human)
        {
            _container.Add(human);
        }

        public T Remove(T human)
        {
            var element = _container.FirstOrDefault(h => h == human);
            if (element != null)
            {
                _container.Remove(element);
                return element;
            }

            throw new NullReferenceException();
        }

        public void Sort()
        {
            _container.Sort();
        }

        public IEnumerator<T> GetEnumerator()
        {
            return _container.GetEnumerator();
        }

        IEnumerator IEnumerable.GetEnumerator()
        {
            return GetEnumerator();
```

```
        }

        #endregion
    }
}
```

Исследуйте исходный код примера:

1 Каким образом в языке C# используется обобщения?

2 Что делает ключевое слово «where» при определении класса HumanContainer?

3 Для какой цели класс Human реализует интерфейс IComparable? Чт описывает данный интерфейс?

4 Объясните назначение интерфейса IEnumerable. Какие методы придется реализовать для того, чтобы воспользоваться данным интерфейсом?

5 Что такое «Итератор». Какой интерфейс описывает свойства и поведение объекта-итератора? Объясните принцип работы итераторов в языке C#.

6 Поясните принцип работы индексатора.

7 Составьте условие задачи для кода примера.

***<u>Содержание отчета:</u>***

1. Номер и тема лабораторной работы.
2. Цель лабораторной работы.
3. Техническое оснащение.
4. При выполнении примеров, необходимо в отчет внести скриншоты готовых программ.
5. Вывод по лабораторной работе.