Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»
Филиал
«Минский радиотехнический колледж»

Учебный предмет
«Конструирование программ и языки программирования»

Инструкция
по выполнению лабораторной работы №24
«Разработка, отладка и испытание программ с обобщениями»

Минск 2024 г.

Лабораторная работа № 24

Тема работы: «Разработка, отладка и испытание программ с обобщениями»

1 Цель работы

Сформировать умение разрабатывать программы с использованием обобщений.

2 Задание

Задания 1 и 2 общее для всех и обязательные для выполнения.

Задание 1

Проанализируйте следующий код, и дайте ответ на вопросы после кода.

```
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
```

```
        }
}
```

Задание 2

Создайте обобщенный класс CollectionType<T>. Определить в классе конструкторы, деструктор, методы добавления и удаления элементов, другие необходимые методы и, если требуется, перегруженные операции. Определить индексаторы и свойства. CollectionType можно реализовать на основе стандартных коллекций (List, Stack, Array и т.д.). Не забывайте про обработку исключений.

3 Оснащение работы

ПК, среда Visual Studio 2019, MSword.

4 Основные теоретические сведения

Чтобы понять, в чем заключаются преимущества использования обобщений, следует выяснить, какие проблемы возникают у программиста без их использования. Вы должны помнить из главы 3, что платформа .NET поддерживает автоматическое преобразование объектов стека и динамической памяти с помощью операций создания объектного образа и восстановления из объектного образа. На первый взгляд, эта возможность кажется не слишком полезной и имеющей, скорее, теоретический, чем практический интерес. Но на самом деле возможность создания объектного образа и восстановления из объектного образа очень полезна тем, что она позволяет интерпретировать все, как System.Object, оставив все заботы о распределении памяти на усмотрение CLR.

Чтобы рассмотреть особенности процесса создания объектного образа, предположим, что мы создали System.Collections.ArrayList для хранения числовых (т.е. размещаемых в стеке) данных. Напомним, что все члены ArrayList обладают прототипами для получения и возвращения типов System.Object. Но вместо того, чтобы заставлять программиста вручную вкладывать размещенное в стеке целое число в соответствующую объектную оболочку, среда выполнения делает это автоматически с помощью операции создания объектного образа.

```
static void Main(string[] args)
{
    // При передаче данных члену, требующему объект, для
    // характеризуемых значениями типов автоматически создается
    // объектный образ.
    ArrayList myInts = new ArrayList();
    myInts.Add(10);
    Console.ReadLine();
}
```

Чтобы восстановить значение из объекта ArrayList, используя индексатор типа, вы должны превратить размещенный в динамической памяти объект в размещенное в стеке целое число, используя операцию преобразования.

```
static void Main(string[] args)
{
...
   // Значение восстанавливается... и снова становится объектом!
   Console.WriteLine("Значение вашего int: {0}",
      (int)myInts[0]);
   Console.ReadLine();
}
```

Для представления операции создания объектного образа в терминах CIL компилятор C# использует блок box. Точно так же операция восстановления из объектного образа преобразуется в CIL-блок unbox. Вот соответствующий CIL-код для показанного выше метода Main() (этот код можно увидеть с помощью ildasm.exe).

```
.method private hidebysig static void Main(string[] args) cil managed
{
...
   box  [mscorlib]System.Int32
   callvirt  instance int32 [mscorlib]
      System.Collections.ArrayList::Add(object)
   pop
   ldstr  "Значение вашего int: {0}"
   ldloc.0
   ldc.i4.0
   callvirt  instance object [mscorlib]
      System.Collections.ArrayList::get_Item(int32)
   unbox  [mscorlib]System.Int32
   ldind.i4
   box  [mscorlib]System.Int32
   call void [mscorlib]System.Console::WriteLine(string, object)
...
}
```

Обратите внимание на то, что перед обращением к ArrayList.Add() размещенное в стеке значение System.Int32 преобразуется в объект, чтобы передать требуемый System.Object. Также заметьте, что при чтении из ArrayList с помощью индексатора типа (что отображается в скрытый метод get_Item()) объект System.Object восстанавливается в System.Int32 только для того, чтобы снова стать объектным образом при передаче методу Console.WriteLine().

Проблемы создания объектных образов и восстановления значений

Операции создания объектных образов и восстановления из них значений очень удобны с точки зрения программиста, но такой упрощенный подход при обмене элементами стека и динамической памяти имеет свои специфические проблемы производительности и не гарантирует типовой безопасности. Чтобы понять проблемы производительности, рассмотрим следующие шаги, которые приходится выполнять при создании объектного образа и восстановлении значения обычного целого числа.

1. Новый объект нужно разместить в управляемой динамической памяти.
2. Значение размещенных в стеке данных нужно записать в соответствующее место в памяти.
3. При восстановлении значения, сохраненного в объекте, размещенном в динамической памяти, это значение нужно снова вернуть в стек.
4. Неиспользуемый объект в управляемой динамической памяти (в конце концов) должен быть уничтожен сборщиком мусора.

В нашей ситуации метод Main() с точки зрения производительности не имеет никаких проблем, но соответствующие проблемы появятся, когда ArrayList будет содержать тысячи целых значений, если вашей программе придется регулярно их обрабатывать.

Теперь рассмотрим проблему отсутствия типовой безопасности в отношении операции восстановления значений из объектного образа. Вы знаете, что для восстановления значения в рамках синтаксиса C# используется оператор преобразования. Но каким будет это преобразование — успешным или неудачным, — выяснится только *в среде выполнения*. При попытке восстановить значение в неправильный тип данных вы получите InvalidCastException.

```
static void Main(string[] args)
{
...
   // Ой!  Исключение времени выполнения!
   Console.WriteLine("Значение вашего int: {0}",
      (short)myInts[0]);
Console.ReadLine();
}
```

В идеальной ситуации компилятор C# должен решать проблемы некорректных операций восстановления из объектного образа во время компиляции, а не в среде выполнения. В связи с этим, в *действительно* идеальной ситуации, можно было бы сохранить типы, характеризуемые значениями, в контейнере, который не требовал бы преобразования в объект. В .NET 2.0 обобщения дают решение именно этих проблем. Однако перед тем как углубиться в детали использования обобщений, давайте посмотрим, как программисты пытались

бороться с этими проблемами в .NET 1.x. с помощью строго типизованных коллекций.

5. Порядок выполнения работы

1. Выделить ключевые моменты задачи.
2. Построить алгоритм и теоретическую объектную модель решения задачи.
3. Запрограммировать полученные алгоритмы и объектную модель.

6. Форма отчета о работе

Лабораторная работа № ___

Номер учебной группы _____

Фамилия, инициалы учащегося _____

Дата выполнения работы _____

Тема работы: _____

Цель работы: _____

Оснащение работы: _____

Результат выполнения работы: _____

7. Контрольные вопросы и задания

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? Что 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. Для чего используется язык интегрированных запросов (Language Integrated Query)?
8. Приведите пример отложенных запросов и тех, что выполняются сразу, в языке LINQ;
9. В чем преимущества отложенных запросов?
10. Каким образом LINQ использует лямбда-выражения?
11. Объясните принцип работы всех LINQ-операций, использованных в примере.

8. Рекомендуемая литература

1. Рихтер, Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# / Дж. Рихтер. СПб. : Изд-во Питер, 2021. 896 с.

2. Прайс, М. Дж. C# 10 и .NET 6. Современная кросс-платформенная разработка / М. Дж. Прайс. СПб : Изд-во Питер, 2023. 848 с.

3. Васильев, А.Н. Программирование на C# для начинающих. Особенности языка / А.Н. Васильев. М. : Эксмо, 2022. 528 с.

4. Фримен, А. ASP.NET Core 3 с примерами на C# для профессионалов / А. Фримен. СПб. : Изд-во Вильямс, 2021. 1184 с.