**Лабораторная работа №17**

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

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

**Технологическое оснащение:** ПК Win 10, MS VS 2019.

**Выполнение работы**

Листинг программы:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Collections;

namespace Задание\_1

{

class Program

{

static void Main(string[] args)

{

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 = 181,

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;

//contaoner1[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: Where");

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} ,TotalWeigth: {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);

}

Console.ReadKey();

}

}

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 Properties

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 DoWord() { }

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

}

Проверка выполнения программы отображена на рисунках 1,2.

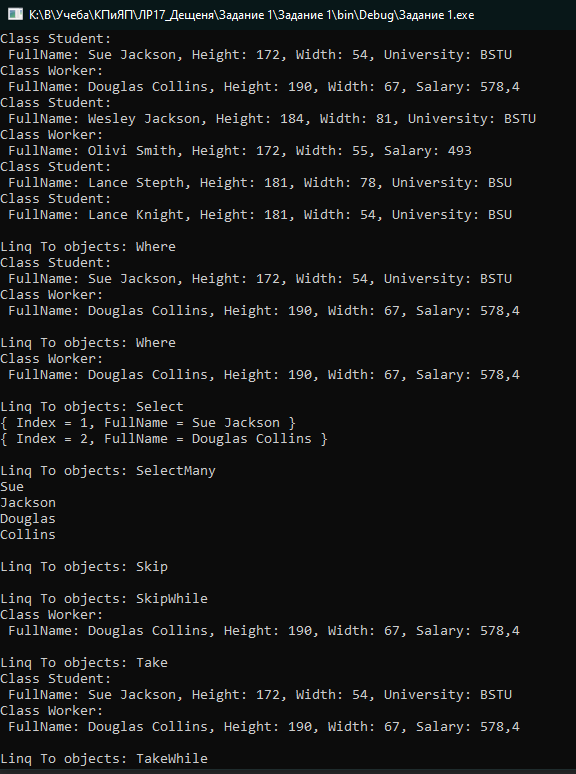


Рисунок 1 – Проверка выполнения программы

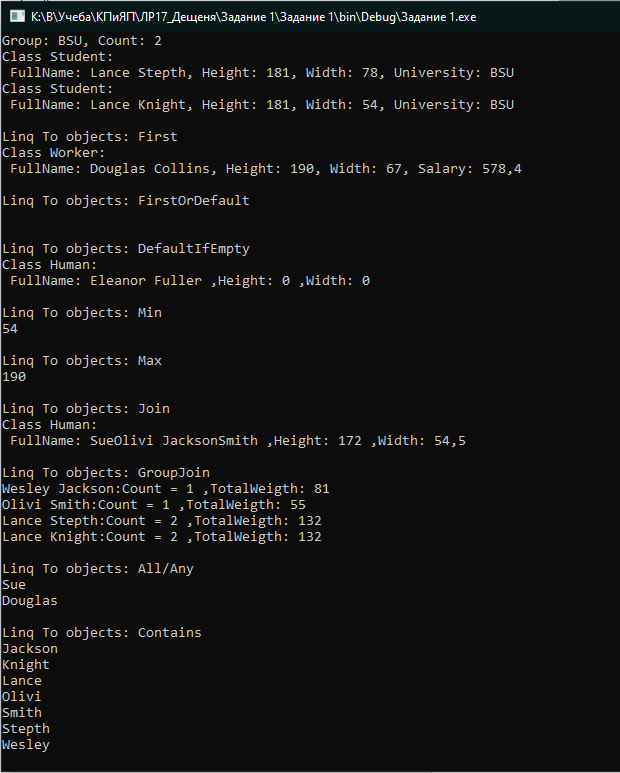


Рисунок 2 – Проверка выполнения программы.

**Вывод:** В ходе лабораторной работы научился разрабатывать программы с использованием обобщений и коллекций.