Московский государственный технический университет и Н.Э. Баумана

Факультет ИУ «Информатика и системы управления»

Кафедра ИУ-5 «Системы обработки информации и управления»

ОТЧЕТ

по лабораторной работе №6

Выполнил:

Студент ИУ5-32Б

Нырков Илья Алексеевич

Проверил:

Доцент Гапанюк Ю. Е.

МОСКВА 2020

**Описание задания** –

**Часть 1. Разработать программу, использующую делегаты.**

(В качестве примера можно использовать проект «Delegates»).

1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Определите делегат, принимающий несколько параметров различных типов и возвращающий значение произвольного типа.
3. Напишите метод, соответствующий данному делегату.
4. Напишите метод, принимающий разработанный Вами делегат, в качестве одного из входным параметров. Осуществите вызов метода, передавая в качестве параметра-делегата:
   * метод, разработанный в пункте 3;
   * лямбда-выражение.
5. Повторите пункт 4, используя вместо разработанного Вами делегата, обобщенный делегат Func< > или Action< >, соответствующий сигнатуре разработанного Вами делегата.

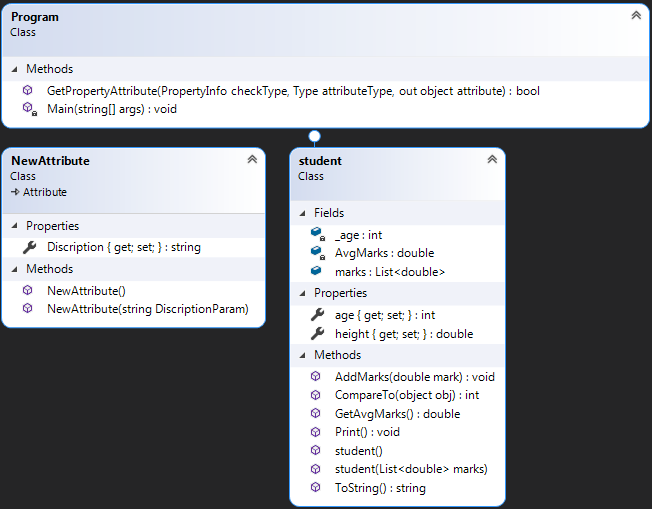
**Часть 2. Разработать программу, реализующую работу с рефлексией.**

(В качестве примера можно использовать проект «Reflection»).

1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Создайте класс, содержащий конструкторы, свойства, методы.
3. С использованием рефлексии выведите информацию о конструкторах, свойствах, методах.
4. Создайте класс атрибута (унаследован от класса System.Attribute).
5. Назначьте атрибут некоторым свойствам классам. Выведите только те свойства, которым назначен атрибут.
6. Вызовите один из методов класса с использованием рефлексии.

**Диаграмма классов:**

**2 Часть**

****

**Листинг кода программы:**

1. **Часть (делегаты)**

using System;

using System.Numerics;

using System.Runtime.Intrinsics;

using System.Transactions;

using System.Collections.Generic;

using System.Linq;

namespace Delegates

{

delegate bool algorithm\_condition<T>(T value);

class Program

{

static bool is\_prime\_number(int value)

{

int j = 0, i = 2;

while ((i\*i <= value) && (j != 1))

{

if (value % i == 0)

j = 1;

i++;

}

return j != 1;

}

static void copy\_if<T>(List<T> arr, List<T> arr2,

algorithm\_condition<T> condition)

{

foreach(var item in arr)

{

if (condition(item))

arr2.Add(item);

}

}

static List<double> two\_variable\_list\_fill(Func<double, double, double> func, int elems\_count)

{

Random rand = new Random();

List<double> list = new List<double>();

for (int i = 0; i < elems\_count; i++)

{

list.Add(func(rand.Next(0, 10), rand.Next(10, 20)));

}

return list;

}

static void remove\_if<T>(ref List<T> arr,

Func<T,bool> condition)

{

List<T> copy = new List<T>();

foreach(var item in arr)

{

if (!condition(item))

copy.Add(item);

}

arr = copy;

}

static void Print\_collection<T>(T arr)

where T : System.Collections.IEnumerable {

foreach(var item in arr)

{

Console.WriteLine(item.ToString());

}

}

static void Fill\_rand\_list(ref List<int> arr, int count,

int left\_lim, int right\_lim){

var rand = new Random();

for (int i = 0; i < count; i++)

{

arr.Add(rand.Next(left\_lim, right\_lim));

}

}

static void Test\_delegates()

{

List<int> arr = new List<int>();

Console.WriteLine("List before functions:\n");

Fill\_rand\_list(ref arr, 20, 0, 10);

Print\_collection(arr);

Console.WriteLine("\nAfter lambda compararsion condition:\n");

remove\_if(ref arr, x => x > 5);

Print\_collection(arr);

Console.WriteLine("\nAfter is\_prime\_number condition:\n");

List<int> copy = new List<int>();

algorithm\_condition<int> condition = is\_prime\_number;

copy\_if(arr, copy, is\_prime\_number);

Print\_collection(copy);

}

static void Main(string[] args)

{

Test\_delegates();

}

}

}

1. **Часть (Рефлексия)**

using System;

using System.Collections.Generic;

using System.Data.SqlTypes;

using System.Reflection;

using System.Security.Cryptography.X509Certificates;

using System.Text;

namespace reflection

{

[AttributeUsage(AttributeTargets.Property, AllowMultiple = false, Inherited = false)]

public class NewAttribute : Attribute

{

public NewAttribute() { }

public NewAttribute(string DiscriptionParam)

{

Discription = DiscriptionParam;

}

public string Discription { get; set; }

}

public class student : IComparable

{

public student() { }

public student(List<double> marks)

{

this.marks = marks;

}

private int \_age;

[NewAttribute("Student age - restrictions:" +

" can't be negative\n")]

public int age{

get

{

return \_age;

}

set

{

if (value < 0)

{

throw new Exception("age can't be negative");

}

else

{

\_age = value;

}

}

}

public double height { get; set; }

public List<double> marks =

new List<double>();

private double AvgMarks;

public double GetAvgMarks()

{

if (marks.Count == 0)

return 0;

double sum = 0;

foreach(int x in marks)

{

sum += x;

}

return sum / marks.Count;

}

public void AddMarks(double mark)

{

marks.Add(mark);

AvgMarks = GetAvgMarks();

}

public override string ToString()

{

StringBuilder str = new StringBuilder();

str.Append("Average score: " + AvgMarks.ToString() + "\n");

foreach (var x in marks)

{

str.Append(x.ToString() + "\n");

}

return str.ToString();

}

public void Print()

{

Console.WriteLine(this.ToString());

}

public int CompareTo(object obj)

{

student obj2 = (student)obj;

if (this.AvgMarks == obj2.AvgMarks)

return 0;

if (this.AvgMarks < obj2.AvgMarks)

return -1;

else

return 1;

}

};

class Program

{

public static bool GetPropertyAttribute(PropertyInfo checkType, Type attributeType, out object attribute)

{

bool Result = false;

attribute = null;

var isAttribute = checkType.GetCustomAttributes(attributeType, false);

if (isAttribute.Length > 0)

{

Result = true;

attribute = isAttribute[0];

}

return Result;

}

static void Main(string[] args)

{

student John = new student();

student Mike = new student();

Type John\_type = John.GetType();

for (int i = 0; i < 5; i++)

{

John.AddMarks(i);

Mike.AddMarks(i + 1);

}

Console.WriteLine("++++++++++++Constructors++++++++");

foreach (var x in John\_type.GetConstructors())

{

Console.WriteLine(x);

}

Console.WriteLine("++++++++++++Properties+++++++++");

foreach (var x in John\_type.GetProperties())

{

Console.WriteLine(x);

}

Console.WriteLine("++++++++++++Methods+++++++++++");

foreach (var x in John\_type.GetMethods())

{

Console.WriteLine(x);

}

Console.WriteLine("++++++++++++Attributed\_Properties+++++++++++++");

foreach (var x in John\_type.GetProperties())

{

if (GetPropertyAttribute(x, typeof(NewAttribute), out object attribute))

{

NewAttribute attr = (NewAttribute)attribute;

Console.WriteLine(x.Name + " - " + attr.Discription);

}

}

Console.WriteLine("++++++++++++Invoked\_method-GetAvgMarks");

object Result = John\_type.InvokeMember("GetAvgMarks", BindingFlags.InvokeMethod, null, John, null);

Console.WriteLine(Result);

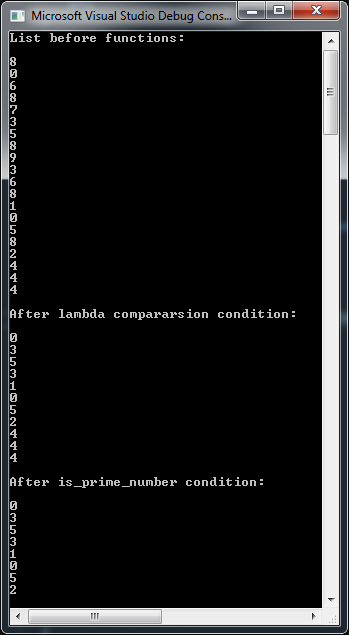
}

}

}

**Скриншоты выполнения программы:**

**1 Часть (делегаты)**



1. **Часть (рефлексия)**

